UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

"DISEÑO E IMPLEMENTACIÓN DE APLICACIONES BASICAS CON EL DSC F2812"

Titulación: **Ingeniería Técnica Industrial, esp. Electrónica Industrial**

Alumno: **Juan Micael Espín Masegosa**

Directores: **D. Jacinto Mª Jiménez Martínez**

Cartagena, Septiembre 2012

Índice

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

0.-ANTECEDENTES Y OBJETIVOS

Uno de los objetivos de este proyecto es facilitar el aprendizaje en la programación de micros en general y del DSP F2812 en concreto.

Con este proyecto también pretendemos crear una base solida con la cual poder desarrollar aplicaciones complejas utilizando todas las posibilidades que nos ofrece el F2812

El aprendizaje de la programación de procesadores digitales de señal puede resultar costoso al principio ,ya que muchos textos de aprendizaje dan por supuesto ciertos conceptos ,por lo tanto con este proyecto se pretende agilizar la primera fase de aprendizaje , el primer contacto con la programación de Dsp, así con una base solida base y sobretodo fácil de entender en la que se explican hasta los conceptos más elementales podemos reducir los tiempos de aprendizaje ,ya que una vez obtenida la base ,que es lo más costoso ,el alumno podrá ir progresando a un ritmo mayor.

En un primer apartado explicaremos todos los recursos del F2812 y seguidamente expondremos ejemplos sencillos y explicados al detalle.

Con ejercicios sencillos conseguimos exponer y explicar los principales recursos del F2812 de forma completa, para poder desarrollar en el futuro aplicaciones más complejas que incluyan estos recursos tales como temporizadores, PWM para generación de señales, convertidor A/D, etc.

Completamos el aprendizaje con una aplicación completa, que incluye todos los recursos explicados con anterioridad.

Todo este material está apoyado con una placa de expansión propia con la cual podremos hacer funcionar nuestras aplicaciones.

El aprendizaje puede resultar más fácil y motivador, si el alumno es capaz de ver funcionando aplicaciones creadas por el mismo, por eso nos apoyaremos en una placa con indicadores leds, swich , pulsadores, altavoz, etc.

Otro de los objetivos de este proyecto es completar la formación de Ingeniería Técnica Industrial especialidad Electrónica Industrial, ya que este proyecto requiere de conocimiento de varias asignaturas tales como electrónica analógica, digital, conocimiento de C, diseño y fabricación de placas tipo PCB y programación de DSP.

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

1.-CARACTERISTICAS Y RECURSOS DSP F2812

En este apartado procederemos a explicar que es un DSP. Explicaremos la estructura y recursos del 28F12, cuyo modelo vamos a utilizar a la hora de programar las aplicaciones básicas.

Explicaremos como funciona los recursos básicos de este micro tal como sus temporizadores, interrupciones, Event Manager o convertidor AD con el fin de comprender las aplicaciones básicas que veremos a continuación y que utilizan dichos recursos.

1.1.- INTRODUCCIÓN AL DSP TMS320F2812

1.1.1.- Introducción al tutorial del C28x

Bienvenido al tutorial de Texas Instrument TMS320F2812. Este material será usado como una guía al estudiante con lecciones y ejercicios dedicados al Controlador Digital de la Señal (DSP) TMS320F2812. Los capítulos le guiarán a través de los elementos de este dispositivo para un aprendizaje de entrenamiento, mientras que para el desarrollo más exhaustivo serán útiles otros recursos como la búsqueda en Internet.

Es necesario un conocimiento básico de la arquitectura del microprocesador y de su programación en lenguaje C. El tutorial está dividido en dos partes: el material de la primera parte podría ser usado para un primer semestre, acompañado por los ejercicios de laboratorio en paralelo. Cada capítulo incluye un detallado manual para ser usado por los estudiantes durante sus sesiones de laboratorio.

Las sesiones experimentales de laboratorio están basadas es la placa eZdsp TMS3202812, el IDE del Code Composer Studio esta ampliado con la eZdsp y con un hardware adicional (La placa adaptadora “Zwickau Adapter Board”) Las copias de esta están permitidas por el autor. Se incluye el esquemático de esta placa, así que puedes construir una para el uso propio sin ningún problema.

La segunda parte de esta serie se introduce más profundamente en los detalles del TMS320F2812. Es una parte más avanzada que puede ser usada para más lecciones opcionales.

1.1.2.- Clasificación de arquitecturas de microcontroladores.

Para empezar hay que explicar algunas palabras clave que son normalmente usadas cuando se habla del control digital y de la electrónica general. El TMS320F2812 pertenece a un grupo de dispositivos que son llamados Controladores Digitales de la Señal (Digital Signal Controller o DSC). En electrónica se usan palabras como “Microprocesador”, “Microcomputador” o “Microcontrolador” para especificar el dispositivo electrónico del cual estamos hablando. Cuando esto es traspasado al procesado digital de la señal, el nombre preferido es”Digital Signal Processors (DSP)”.

Para comenzar se introducirán algunas definiciones:

- Microprocesador (uP) (Características en figura 1.1)
- Microcomputador
- Microcontrolador (uC)
- Procesador Digital de la Señal (DSP)
- Controlador Digital de la Señal (DSC)

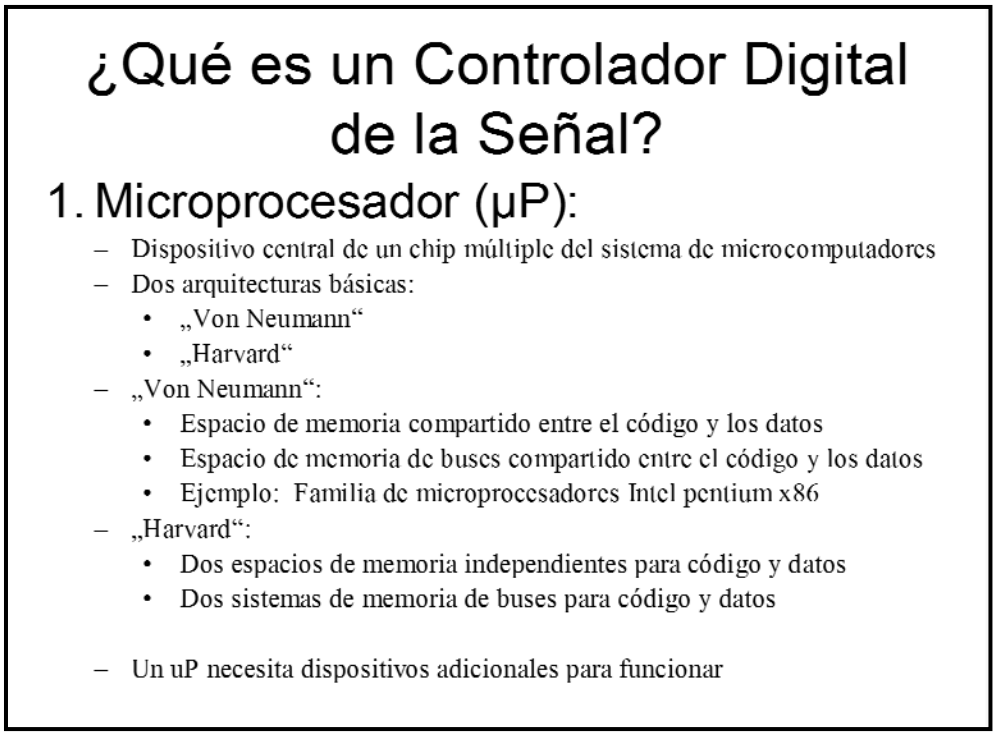


Figura 1.1: Características de un microprocesador.

Los Microprocesadores están basados en un procedimiento secuencial simple, que básicamente podría ser el siguiente:

- Leer de la memoria de programa la siguiente instrucción,
- Decodificar la instrucción,
- Leer los operandos opcionales de la memoria de datos,
- Ejecutar la instrucción devolviendo un resultado.

Esta serie de secuencias se ejecutan de manera infinita. Para usar un Microprocesador se debe añadir una memoria adicional externa al Microprocesador.

1.1.3.- Intel 80x86: Un microprocesador típico

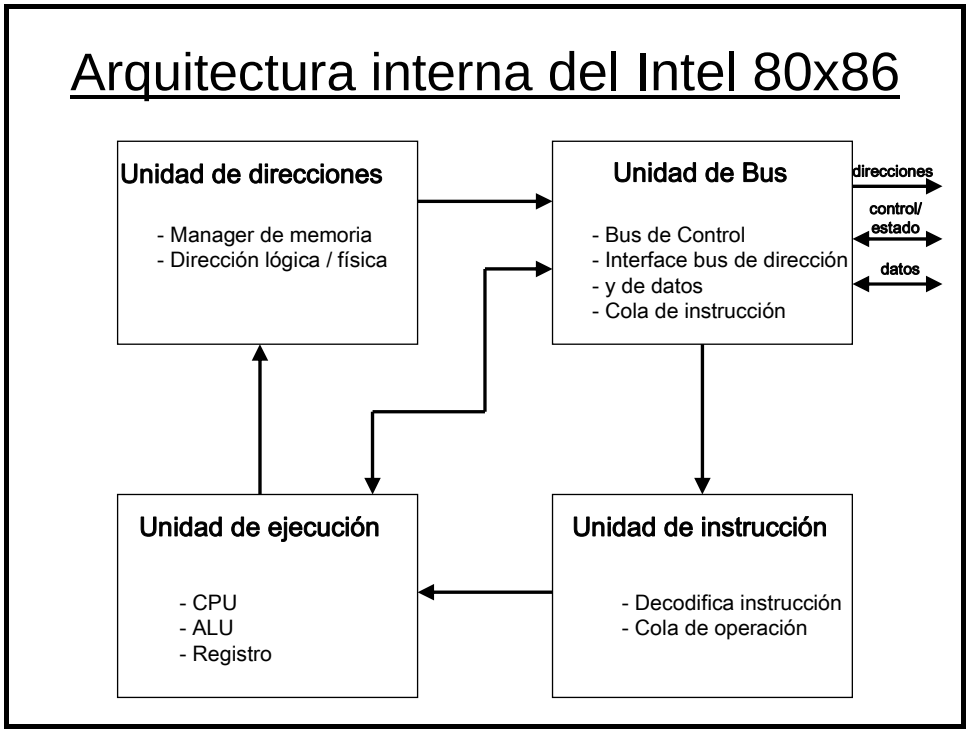


Figura 1.2: Arquitectura interna del Intel 80x86

El intel 8086 puede ser considerado como el veterano de todos los microprocesadores. Dentro de este procesador hay cuatro unidades (Figura 1.2) que toman la secuencia de estados. La **unidad de bus** es responsable del direccionamiento de la memoria externa usando un grupo de señales de direcciones, líneas de datos bidireccionales y señales de control y estado. Su propósito es el de rellenar la primera pipeline, llamada cola de instrucción (instruction queue), la cual es controlada por la unidad de ejecución y la unidad de dirección (Ejecution-Unit and the Address-Unit).

La **unidad de instrucción** lee la siguiente instrucción fuera de la cola de instrucciones la descifra y llena una segunda cola, la cola de operación con las siguientes operaciones internas que deben ser usadas por la unidad de ejecución.

La **unidad de ejecución** hace el trabajo 'real', ejecuta operaciones o llamadas a la unidad de Bus para leer y operaciones opcionales desde la memoria.

Cada vez que una instrucción se completa, la unidad de ejecución fuerza a la unidad de direccionamiento a generar la dirección de la siguiente instrucción. Si esta instrucción esta cargada actualmente en la cola de instrucciones la velocidad de la operación se incrementa. Este principio es el llamado "caché".

1.1.4.- El PC: un microcomputador

Un PC es la agrupación de un microprocesador junto con una memoria y una serie de periféricos. (Figura 1.3).

Cuando se añaden dispositivos externos al microprocesador (Figura 1.4), queda terminada la configuración del sistema de ordenador. Hay que añadir una memoria externa para las instrucciones (código) y una de datos para ser utilizados. Además se necesitará usar conexiones cortas al exterior del sistema. En general, todo esto se agrupa en entradas/salidas digitales y entradas/salidas analógicas.

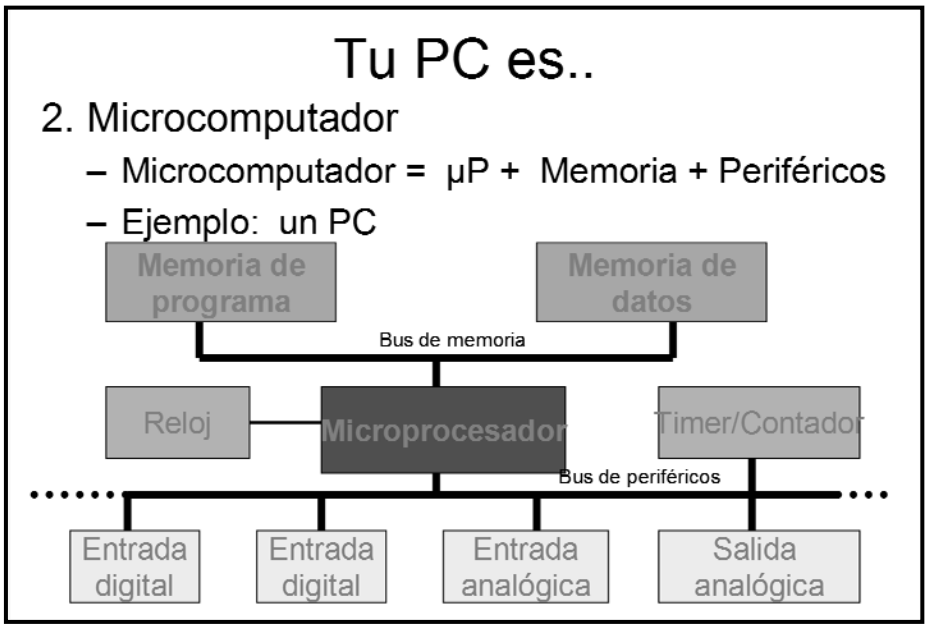


Figura 1.3: Componentes de un PC

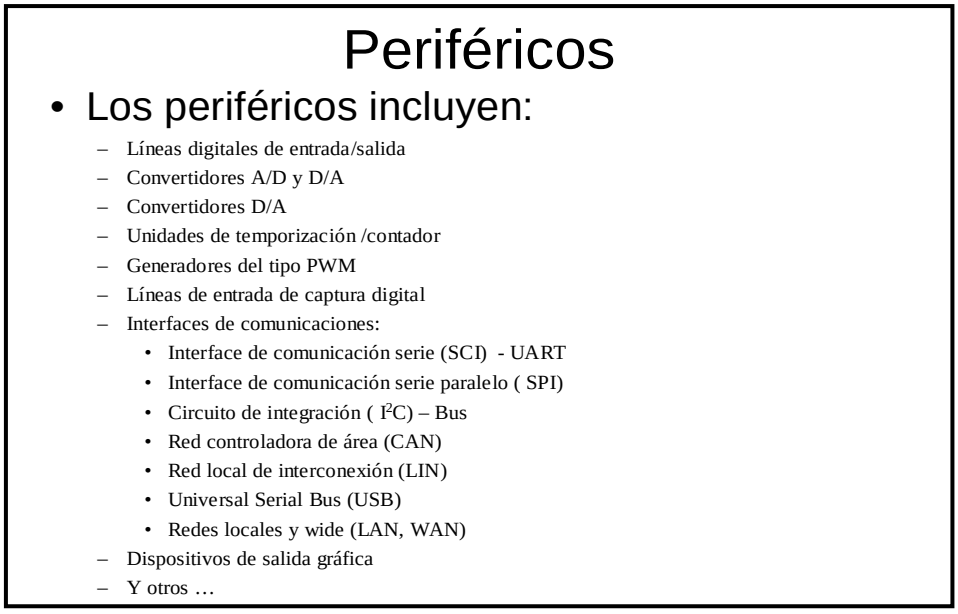


Figura 1.4: Periféricos de un PC

1.1.5. - El microcontrolador: un chip computador simple

Dado que la tecnología avanza, la industria de silicio puede fabricar cualquier cosa que sea necesaria para los microcomputadores dentro de una pieza de silicio, y así se puede hacer el microcontrolador ("µC") (Figura 1.5). Por supuesto nadie pretende incluir unos periféricos simples que estén activos o funcionales dentro de un chip simple- porque nadie podría comprar este chip "monstruo". Por el contrario, los ingenieros demandan microcontroladores que valgan para la mayoría de las aplicaciones. Esto nos permite un gran número de familias de microcontroladores con unidades internas totalmente diferentes, diferentes sets de instrucciones, diferentes números de periféricos y espacios de memoria interna. No se pregunta por un microcontrolador con una capacidad de memoria interna de código de 16 Mbytes, si la aplicación ocupará fácilmente los 64 Kbytes.

Hoy en día los microcontroladores están construidos industrialmente para que sean compatibles con el mercado. Prueba esto, ¿Cuántos microcontroladores hay en tu casa? El problema es que no se pude apreciar desde fuera el microcontrolador. Esta es la razón por la que se suele llamar "embeded". Un producto sofisticado como los modernos equipamientos de coche pueden usar hasta 80 microcontroladores para ejecutar las nuevas funciones electrónicas del coche como ABS, ESP, ACC, etc. Por otra parte un dispositivo simple como un aspirador esta equipado con un microcontrolador para el control de la velocidad y el estado de llenado del limpiador.

Los microcontroladores están disponibles como 4, 8, 16, 32 o 64 bits, el número se debe al número de bits de cada operación que está siendo procesada en paralelo. Si cada microcontrolador es del tipo 32 bits, la memoria de datos interna esta conectada a la unidad del núcleo con 32 líneas de señal.

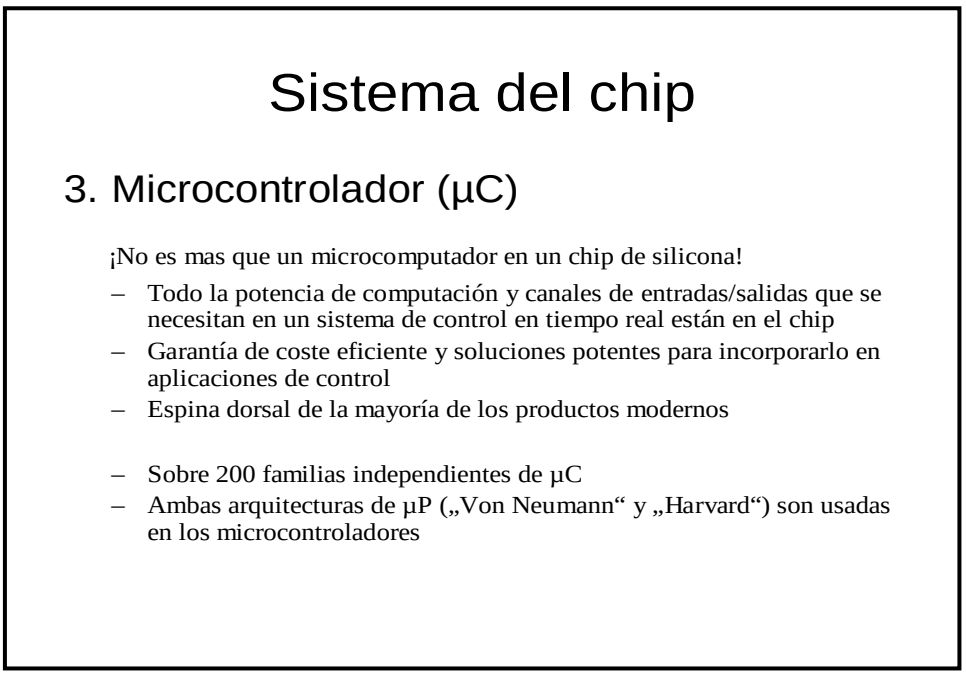


Figura 1.5: Características de un microcontrolador

1.1.6. - El Procesador Digital de la Señal

El procesador digital de la señal es un dispositivo específico que esta diseñado para manipular las operaciones matemáticas más típicas (Figura 1.6 y 1.7) que normalmente se usan con los sensores digitales. El objetivo es hacer el proceso lo más rápido posible como sea posible para generar una señal de salida con el nuevo valor en "tiempo real".

Figura 1.6: Definición de procesador digital de la señal.

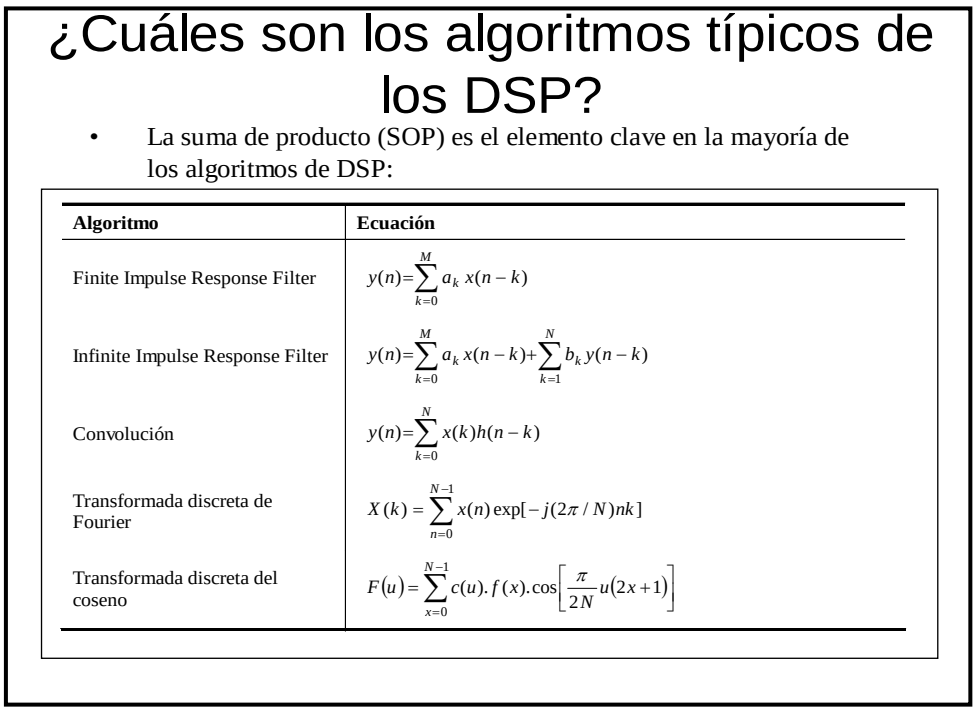


Figura 1.7: Algoritmos de funcionamiento de un DSP

1.1.7. - La ecuación “suma de productos”

No se va a entrar en detalles de la teoría del procesador digital de la señal puesto que no es un tema sencillo. Para comenzar, la aproximación mas utilizada en el procesado digital de la señal es la llamada suma de productos (SOP) (Figura 1.8). Un valor nuevo de “y” es calculado como una suma parcial de productos. Dependiendo del tipo de datos de los arrays podemos resolver esta ecuación mediante punto flotante o con integrales matemáticas.

Mirando un poco mas en detalle en de las tareas que se necesitan para resolver mediante un procesador estándar, se pueden distinguir 10 etapas (Figura 1.9). El doble para la secuencia natural de este tipo de procesadores, este puede hacer los 10 pasos en uno solo. Esto consume una potencia considerable. En este ejemplo, el procesador debe hacer un bucle entre el paso 3 y el paso 10, un total de 4 ciclos de reloj. Para un Procesador Digital de la Señal, esta operación fuerza al procesador estándar a consumir mas potencia.

Estos pasos son trascritos a lenguaje maquina para que puedan ser entendidos por el microcontrolador. (Figura 1.10)

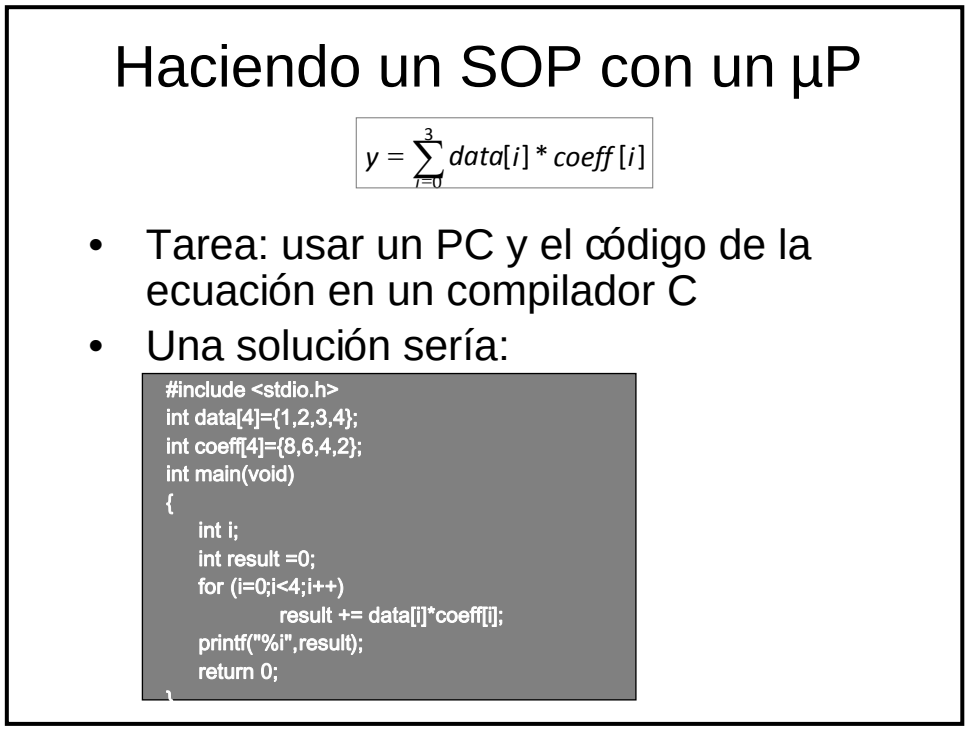


Figura 1.8: Realizar una suma de productos usando un microprocesador

6 Operaciones básicas con un SOP

$$y = \sum_{i=0}^{3} data[i] * coeff[i]$$

- Pasos que realiza un microprocesador
  1. **Fijar un pointer1 para se ñalar los datos**
  2. **Fijar un segundo pointer para se ñalar los coeficientes [0]**
  3. **Datos leidos [i] en el núcleo**
  4. **Leer los coeficientes [i] en el n úcleo**
  5. **Multiplicar data[i]*coeff[i]**
  6. **A ñadir el ultimo producto de los anteriores**
  7. **Modificar el Pointer1**
  8. **Modificar el Pointer2**
  9. **Incrementar I;**
  10. **si i<3 , despu és ir de nuevo al paso 3 y continuar**

- **Los pasos del 3 al 8 son llamado "las 6 operaciones basicas del DSP "**
- **¡Un DSP puede ejecutar los 6 pasos en un solo ciclo!**

Figura 1.9: Pasos que sigue un microprocesador para realizar un SOP

Lenguaje máquina del SOP para µP

```
Dirección       M-Código                          Instrucción en ensamblador
10:       for (i=0;i<4;i++)
00411960        C7 45 FC 00 00 00 00      mov     dword ptr [i],0
00411967        EB 09                     jmp     main+22h (411972h)
00411969        8B 45 FC                  mov     eax,dword ptr [i]
0041196C        83 C0 01                  add     eax,1
0041196F        89 45 FC                  mov     dword ptr [i],eax
00411972        83 7D FC 04       cmp     dword ptr [i],4
00411976        7D 1F                     jge     main+47h (411997h)
11:       result += data[i]*coeff[i];
00411978        8B 45 FC                  mov     eax,dword ptr [i]
0041197B        8B 4D FC                  mov     ecx,dword ptr [i]
0041197E        8B 14 85 40 5B 42 00      mov     edx,dword ptr[eax*4+425B40h]
00411985        0F AF 14 8D 50 5B 42 00 imul  edx,dword ptr[ecx*4+425B50h]
0041198D        8B 45 F8                  mov     eax,dword ptr [result]
00411990        03 C2                     add     eax,edx
00411992        89 45 F8                  mov     dword ptr [result],eax
00411995        EB D2                     jmp     main+19h (411969h)
```

Figura 1.10: Lenguaje maquina para realizar un SOP por un microcontrolador

1.1.8. - Un SOP ejecutado mediante un DSP

Si se realiza la suma de productos en un DSP del tipo coma fija el código ANSI-C hace lo mismo que un procesador estándar. La diferencia esta tras la compilación y pasar a ensamblador. Cuando se compara la figura 1.10 con la figura 1.12 se observa la reducción en la memoria empleada y en el número de ciclos de ejecución. La conclusión a la que se llega es que un DSP es mucho mas apropiado que un microcontrolador para calcular una operación de este tipo en tiempo real.

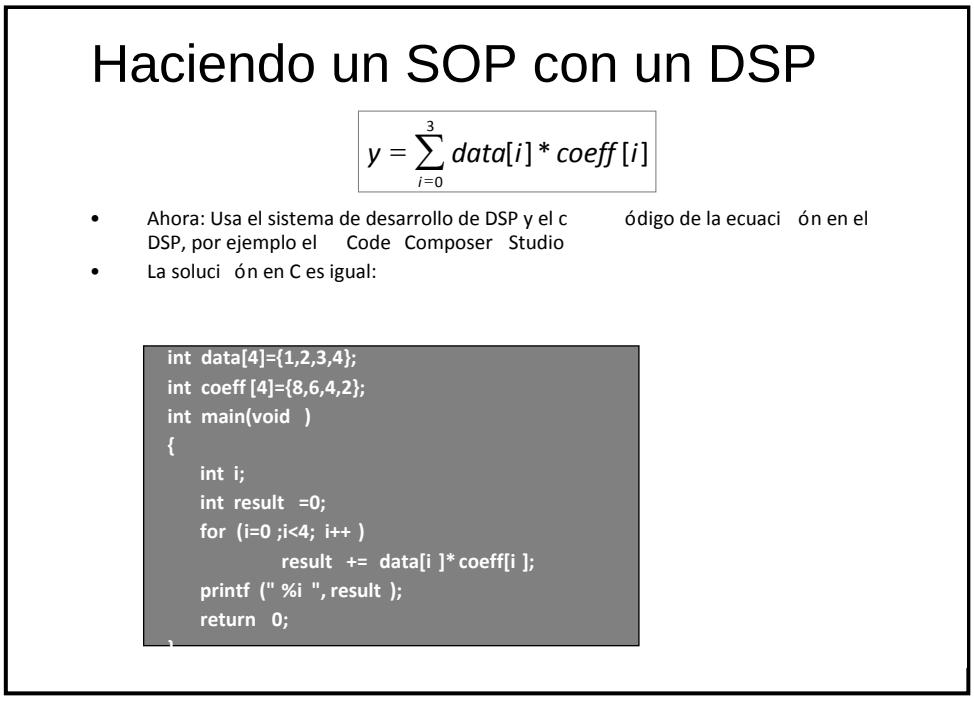


Figura 1.11: Código para realizar un SOP por un DSP

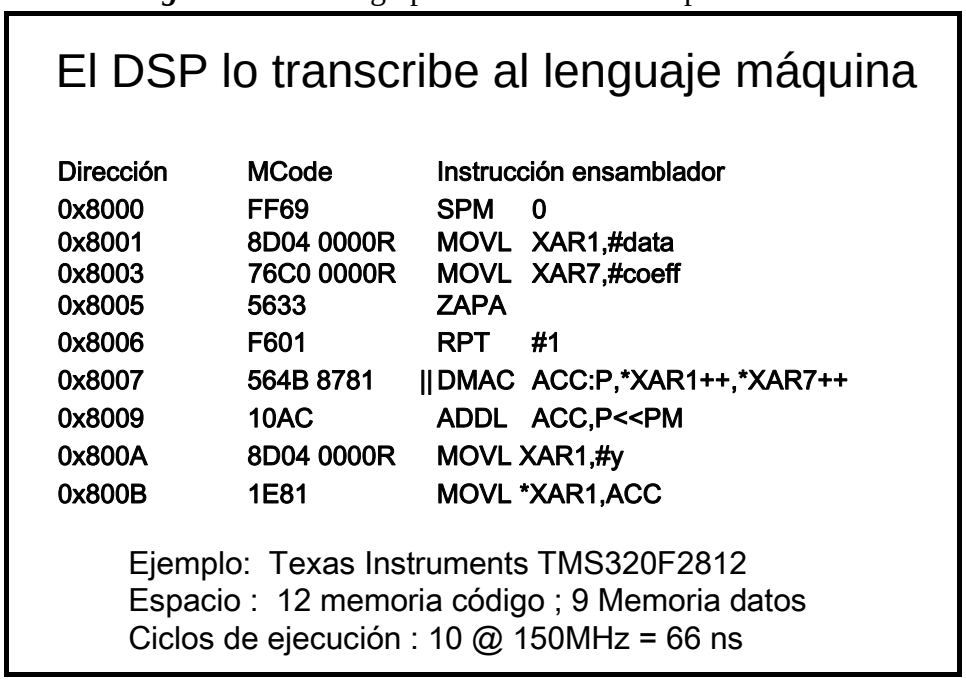


Figura 1.12: Trascripción a código maquina del lenguaje C.

1.1.9.- El Controlador Digital de la Señal

Finalmente, el Controlador Digital de la Señal (DSC) es un nuevo tipo de microcontrolador, donde la potencia de procesamiento es mucho mayor a la de los DSP – un chip simple combinado con el poder de un computador del procesador digital de la señal y con todas las posibilidades de conectar periféricos de un sistema de chip simple.

Para sistemas de control avanzado de tiempo real con un alto numero de operaciones matemáticas, el DSC es la mejor elección.

Este tutorial se basa en el TMS320F2812 de Texas Instrument, un Controlador Digital de la Señal (DSC) de 32 bit en coma fija.

Recuerda que: microcomputador es un microprocesador con memoria y periféricos. Y microcontrolador es un microcomputador en un solo chip.

DSC es la combinación de un microcomputador con un núcleo DSP. Une toda la potencia de un DSP con la integración en un único dispositivo de memoria y periféricos.

Es la solución óptima para sistemas de control integrados donde es necesario una gran cantidad de operaciones.

1.1.10.- El mercado de los DSP

Hay varios fabricantes que producen DSPs y DSCs. Como se puede ver en la figura 1.13, Texas Instrument es el líder absoluto en esta área.

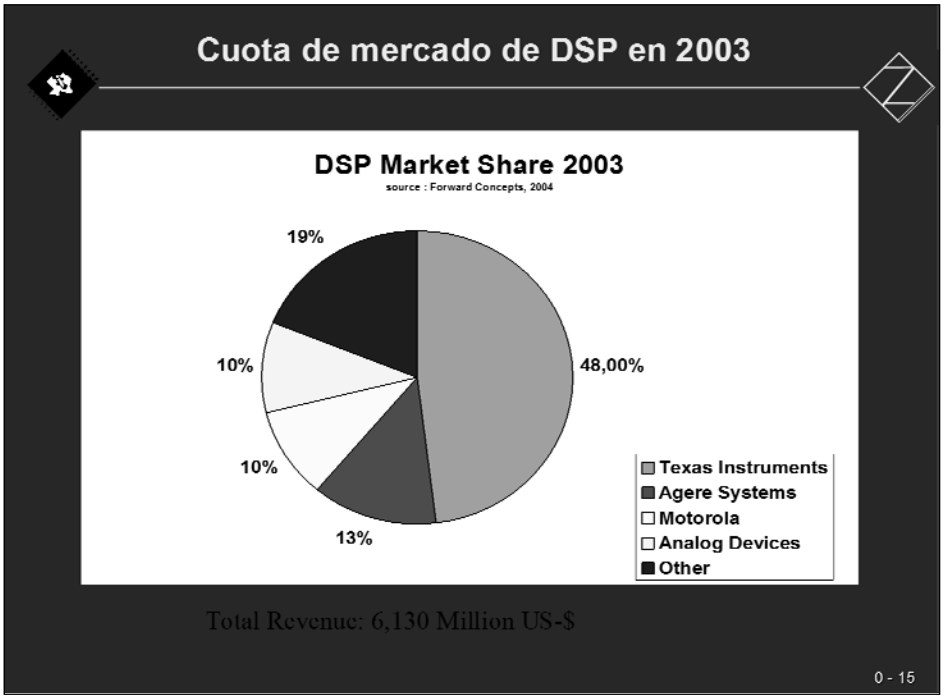


Figura 1.13: Diagrama circular de ventas por marca.

1.1.11.- Las familias de DSP de Texas Instrument

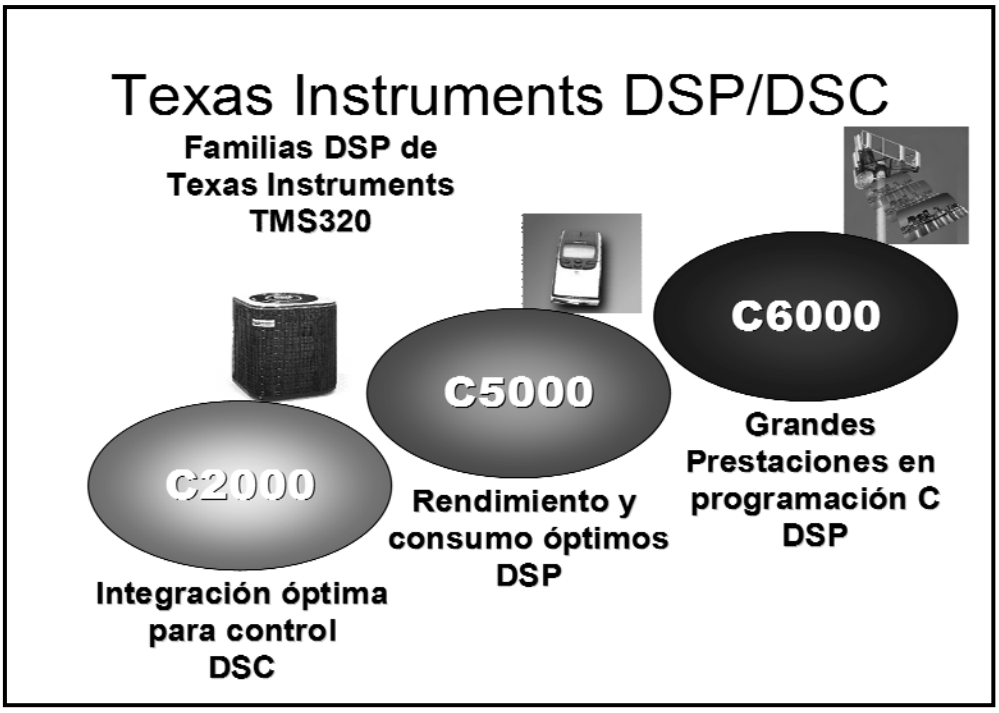


Figura 1.14: Clasificación por familias de los DSP

Los DSP´s de Texas Instrument se pueden diferenciar en tres familias de dispositivos, llamados C2000, C5000 y C6000 (Figura 1.14):

El C6000 es la serie más potente de DSP en cuanto a potencia de computación. Hay de coma flotante y coma fija. Las aplicaciones son para procesamiento de imágenes, audio, servicios multimedia, estaciones base para comunicación gíreles etc.

La familia C5000 es usada para sistemas de móviles con un consumo muy eficiente. La principal aplicación de esta familia es la tecnología de los teléfonos móviles.

La familia C2000 es dedicada al Control Digital de la Señal (DSC), como se comentó en las primeras diapositivas y es una solución muy potente para aplicaciones de control en tiempo real.

En la figura 1.15 resume las principales áreas de aplicaciones para las 3 familias de DSP´s de Texas Instruments.

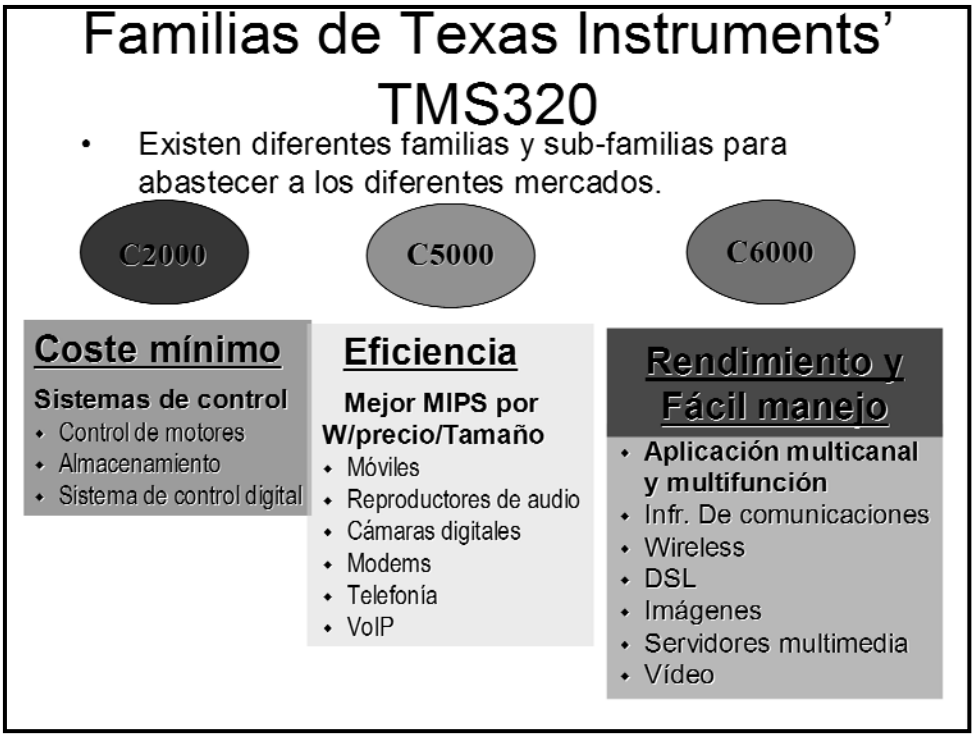


Figura 1.15: Áreas de aplicación de las familias de DSP

Para la familia C2000 se distinguen entre 3 grupos de DSC´s de fixed-point: el grupo de 16 bit, llamado C24x, el grupo de 32 bits, llamado C28x y el grupo que incluye memoria flash, llamado F28xx (Figura 1.16).

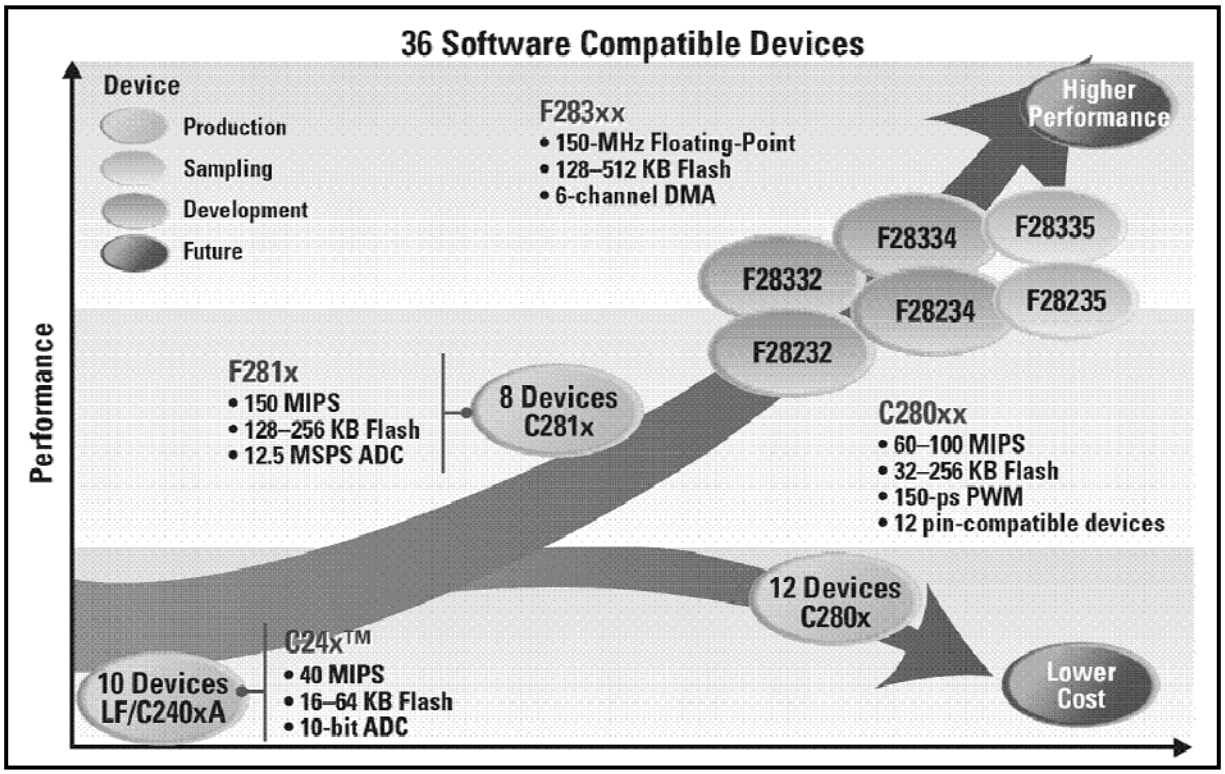


Figura 1.16: Evolución de las familias de DSP

1.1.12.- Evolución del F28x

En la figura 1.17 se ven los avances de la familia C28x asi como la amplia versatilidad del mismo.

Figura 1.17: Aplicaciones de la familia C28XX.

Las especificaciones técnicas de cada uno de los microcontroladores de la familia C2000 quedan recogidas en la tabla 1.18

TI C2000: Lista de usos

	F2812	F2810	LF2407A	LF2406A	LF2403A	LF2402A	LF2401A	LC2406A	LC2404A	LC2402A	LC2401A	F283	F241	F240	C242
CPU	32bit	32 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16bit	16bit
MIPS	150	150	40	40	40	40	40	40	40	40	40	20	20	20	20
RAM (words)	18K	18K	2.5K	2.5K	1.0K	1.0K	1.0K	2.5K	1.5K	544	1.0K	544	544	544	544
ROM (words)								32K	16K	6K	8K				4K
Flash (words)	128K	64K	32K	32K	16K	8K	8K					8K	8K	16K	
BootROM (words)	4K	4K	256	256	256	256	256								
Event Manager CAP/QEP	6/6	6/6	6/4	6/4	3/2	3/2	1/0	6/4	6/4	3/2	1/0	3/2	3/2	4/2	3/2
PWM (CMP)	16	16	16	16	8	8	7	16	16	8	7	8	8	12	8
TIMER	7	7	4	4	2	2	2	4	4	2	2	2	2	3	2
ADC Resolution	12-bit	12-bit	10-bit	10-bit	10-bit	10-bit	10-bit	10 bit	10-bit	10-bit	10-bit	10-bit	10-bit	10-bit	10-bit
# ofChan	16	16	16	16	8	8	5	16	16	8	5	8	8	16	8
Conv time	200ns	200ns	500ns	500ns	500ns	500ns	500ns	375ns	375ns	425ns	500ns	900ns	900ns	6.1us	900ns
McBSP	✓	✓													
EXMIF	✓		✓									✓		✓	
Watch Dog	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SPI	✓	✓	✓	✓	✓			✓	✓			✓	✓	✓	
SCI (UART)	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1
CAN	✓	✓	✓	✓	✓			✓				✓			
Volts (V)	1.8 core 3.3 I/O	1.8core 3.3 I/O	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	5.0	5.0	5.0	5.0
# I/O	56	56	41	41	21	21	13	41	41	21	13	32	26	28	26
Package	176LQFP 179u*BGA	128LQFP	144LQFP	100LQFP	64LQFP	64PQFP	32LQFP	100LQFP	100LQFP	64PQFP	32LQFP	144LQFP	64PQFP 68PLCC	132PQFP	64PQFP 68PLCC

Tabla 1.18: Tabla de utilidades de cada modelo de la familia C2000

1.2.- ARQUITECTURA DEL DSP TMS320F2812

1.2.1.- Introducción

Puesto que un procesador digital de señal es capaz de ejecutar seis operaciones básicas en un solo ciclo de la instrucción, la arquitectura interna del TMS320F2812 debe reflejar esta rapidez de una cierta manera. Entre otras cosas, se discutirán las partes siguientes de la arquitectura del DSP:

- Estructura interna del bus
- CPU
- Multiplicador Hardware, Unidad Aritmética Lógica y desplazador Hardware
- Estructura Del Registro
- Mapa de Memoria

1.2.2.- El diagrama de bloques del TMS320F2812

El diagrama de bloque TMS320F2812 se puede dividir en 4 bloques funcionales (Figura 2.1):

- Bus interno y externo
- Unidad central de proceso (CPU)
- Memoria
- Periféricos

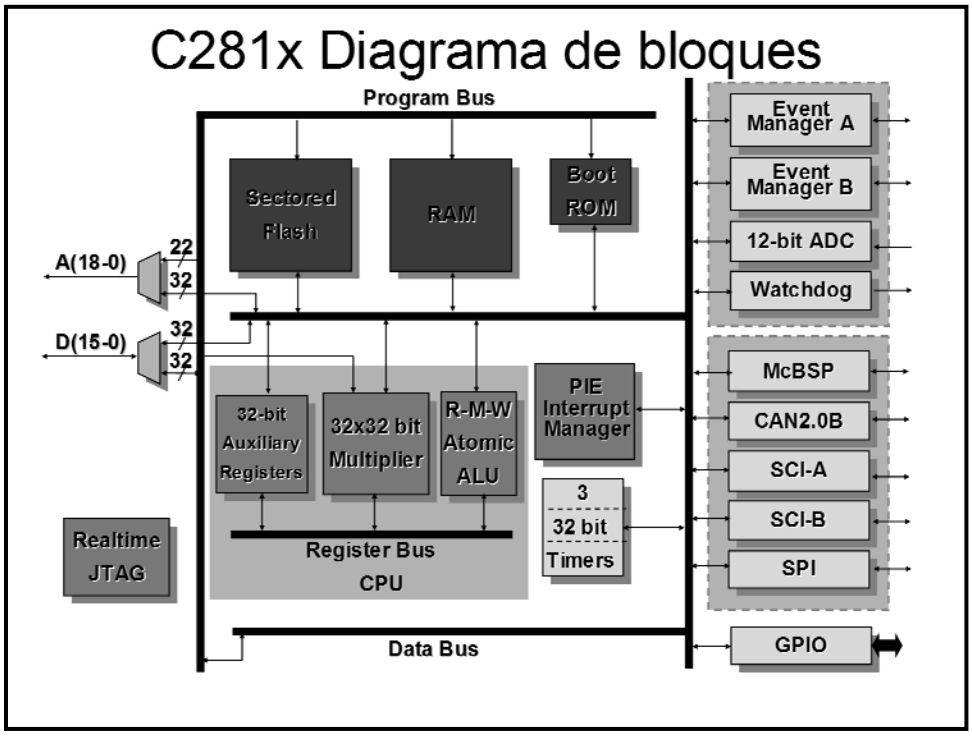


Figura 2.1: Diagrama de bloques del C281X

Para recuperar dos operandos de la memoria en la unidad central de proceso en un solo ciclo de reloj el F2812 se equipa de dos sistemas independientes del BUS – bus de programa y bus de datos-. Este tipo de arquitectura se le llama una "Arquitectura Harvard". Debido a la capacidad del F2812 de leer operandos no solamente de la memoria de los datos sino también de la memoria del programa, Texas Instruments llama este dispositivo una " Arquitectura Harvard modificada".

En el lado izquierdo de la figura 2.1 se pueden ver dos bloques del multiplexor para los datos (D15-D0) y la dirección (A18-A0). Es un interfaz para conectar los dispositivos externos con el F2812. Ahora obsérvese que (1) la anchura del Bus de datos externo es solamente 16 bits y que (2), no se puede tener acceso a los datos externos del bus de programa y a los datos del bus de datos en el mismo tiempo. Comparado a un solo ciclo para el acceso interno a dos operandos de 32-bits, el acceso a memoria externa realiza al menos 4 ciclos.

1.2.3.- La CPU del F2812

La CPU de F2812 puede ejecutar la mayoría de las instrucciones matemáticas gracias a los registros internos que posee y a una gama de instrucciones que son utilizadas comúnmente por los microcontroladores. La arquitectura también incluye modos de direccionamiento especiales, que facilitan la obtención de un código muy compacto tanto en ensamblador como programación en C.

El F2812 es eficiente en tareas matemáticas de procesado así como en las tareas del control de sistema que son manejadas típicamente por los dispositivos del microcontrolador (Figura 2.2).

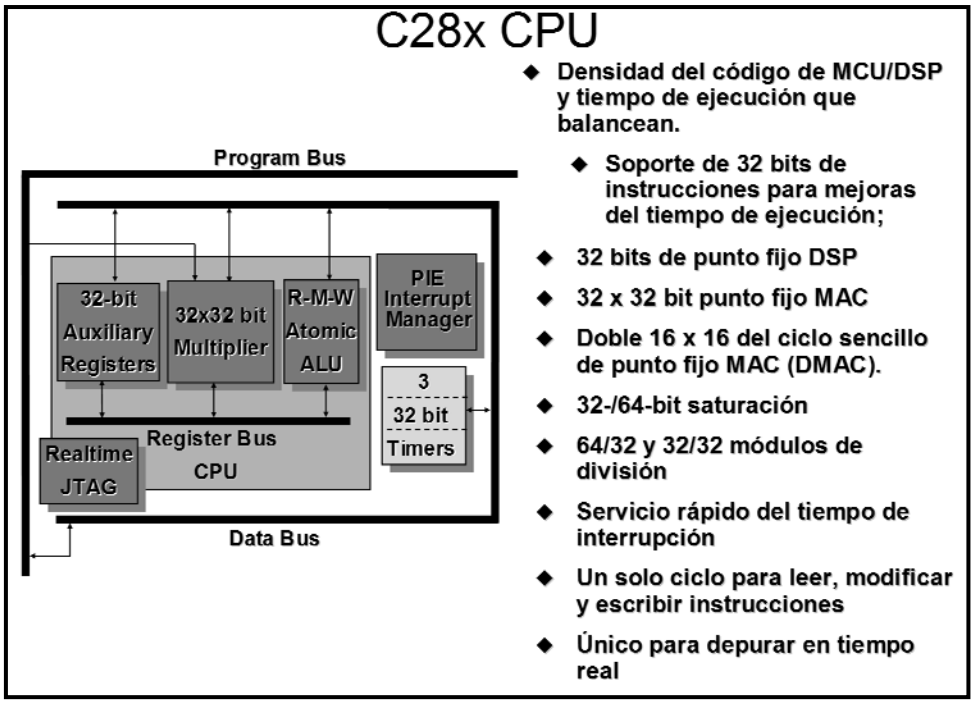


Figura 2.2: Diagrama de bloques de la CPU del C28XX

Incluye tres contadores o temporizadores de 32-bits que pueden ser utilizados para los propósitos generales de la sincronización o para generar los períodos del hardware para los sistemas operativos en tiempo real. También cuenta con un gestor de interrupciones del periférico de extensión de interrupciones (PIE) y permite una respuesta rápida a una interrupción de varias fuentes de señales y de acontecimientos externos e internos.

El Hardware multiplicador y una unidad aritmético lógica de 32-bits (ALU) se pueden utilizar en paralelo para ejecutar una multiplicación y una suma simultáneamente. El banco auxiliar de registros está equipado con su propia unidad aritmética lógica (ARAU) también usado en paralelo para mejorar el indicador aritmético.

El interfaz JTAG es una herramienta para facilitar el intercambio de datos en tiempo real entre el DSC y un host durante la fase de eliminación de errores del desarrollo del proyecto. Es posible observar variables en tiempo real a lavez que el código está funcionando, sin ningún retraso del código de control.

1.2.4.- Unidad Matemática del F2812

Los Multiplicadores y Acumuladores (MAC) de 32 bits del F2812 y sus 64 bits internos (Figura 4.3), permiten a este DSC manejar de manera eficiente los problemas de resolución numéricos de más alto nivel que de otra manera exigirían una solución de procesador de coma flotante más cara. También existe la posibilidad de hacer funcionar dos 16 x 16 bites que multipliquen y acumulen instrucciones simultáneamente (MAC Dual) (DMAC).

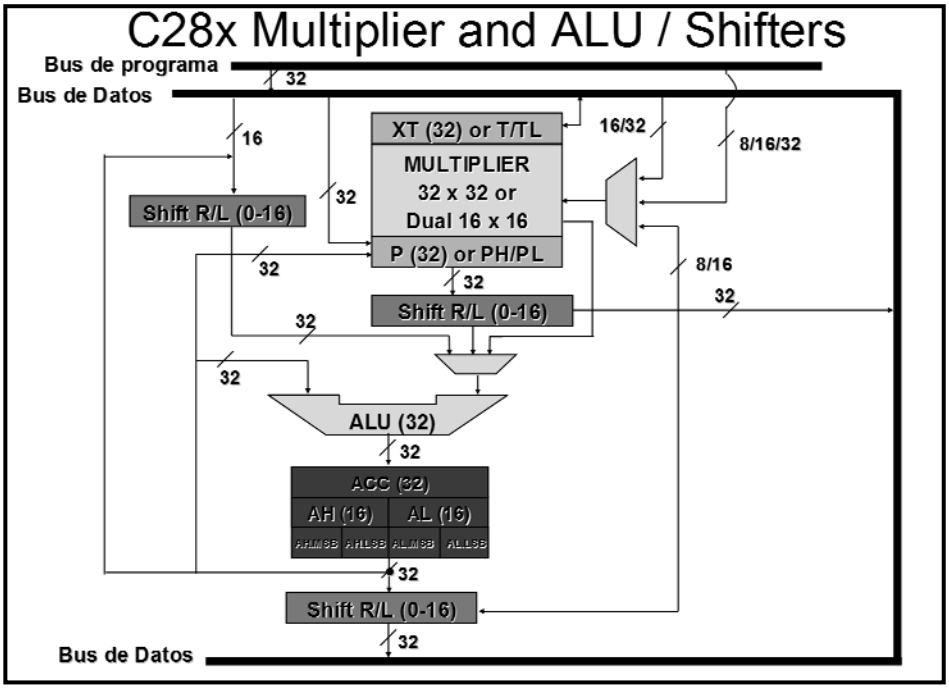


Figura 2.3: Multiplicador y unidad aritmético-lógica del C28XX

La multiplicación utiliza el registro de XT para llevar a cabo el primer operando y para multiplicarlo por un segundo operando que se cargue en la memoria. Si XT se carga de una posición de memoria de datos y el segundo operando se carga de una posición de memoria del programa, la operación de multiplicación puede ser realizada en un solo ciclo.

El resultado de una multiplicación se puede cargar en el registro P (producto) o directamente en el acumulador (ACC). Pregunta, ¿si usted multiplica 32 x 32 bits de números, cuál es el tamaño del resultado? Respuesta: 64-bit. El sistema de instrucción F2812 incluye dos grupos de operaciones multiplicadoras para cargar ambas mitades del resultado en P y el ACC.

Tres registros de desplazamiento del hardware se pueden utilizar en paralelo a otras unidades del hardware de la CPU. Los registros de desplazamiento son utilizados generalmente para escalar números intermedios en un lazo de control en tiempo real o para multiplicar o dividir por 2^n.

La unidad aritmético lógica (ALU) se encarga de hacer el resto de la matemática. El primer operando es siempre el contenido del acumulador (ACC) o de una parte de él. El segundo operando para una operación se carga de la memoria de los datos, de memoria del programa, del registro de P o directamente de la unidad multiplicadora.

1.2.5.- Acceso a la memoria de datos

Dos métodos básicos están disponibles para tener acceso a posiciones de memoria de los datos (Figura 2.4):

- Modo de direccionamiento directo
- Modo de direccionamiento indirecto

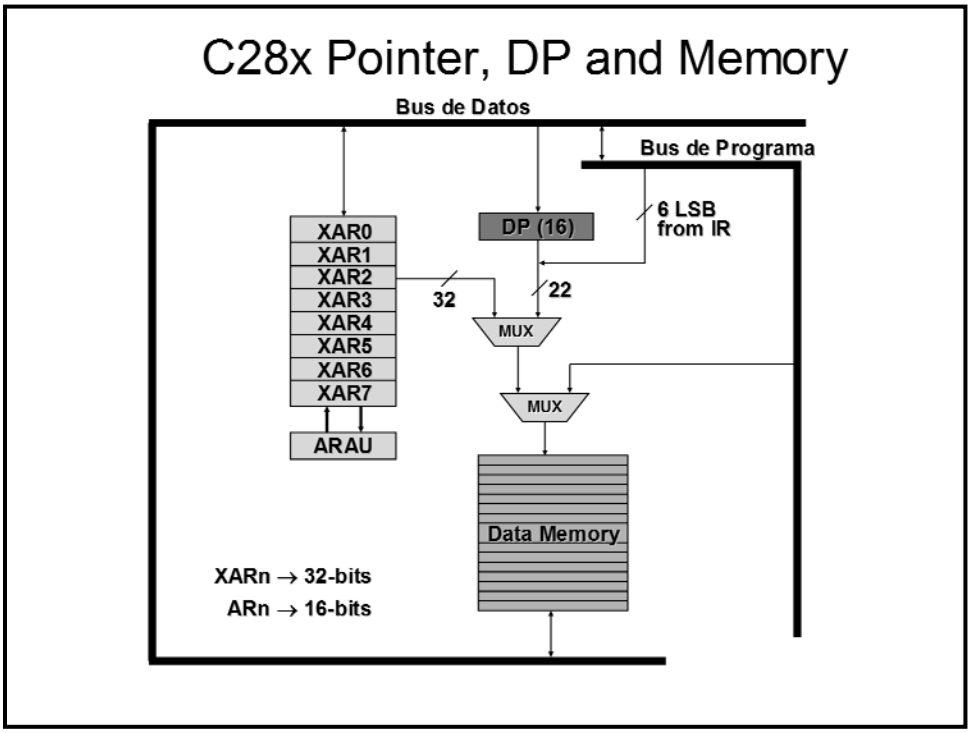


Figura 2.4: Direccionamiento del C28XX

El modo de direccionamiento directo genera la dirección 22-bit para un acceso de memoria a partir de dos fuentes, un registro 16-bit "página de los datos (DP)" para los 16 bits más altos y otros 6 bits tomados de la instrucción. Ventaja: Una vez que se fije el DP, podemos tener acceso a cualquier localización de la página seleccionada, en cualquier orden. Desventaja: Si el código necesita tener acceso a otra página, el DP debe ajustarse primero.

El modo de direccionamiento indirecto utiliza uno de ocho registros 32-bit de XARn para llevar a cabo la dirección 32-bit del operando. Ventaja: Con la ayuda del ARAU, la aritmética del puntero está disponible en el mismo ciclo en el cual se hace un acceso a una posición de memoria de los datos. Desventaja: Un acceso al azar a los datos la memoria necesita una nueva disposición del registro del puntero.

La unidad aritmética del registro auxiliar (ARAU) puede realizar manipulaciones del puntero en el mismo ciclo de reloj que se hace el acceso a una posición de memoria de los datos. Las opciones para el ARAU son: el post-incremento, el pre-decremento, la adición del índice y la substracción.

1.2.6.- Estructura Interna del Bus

Como muchos dispositivos de DSP, los Buses múltiples se utilizan para mover datos entre las posiciones de memoria, unidades periféricas y la CPU. La arquitectura del Bus de la memoria F2812 contiene (Figura 2.5):

- Un programa para leer el Bus (línea de dirección de 22 bits y 32 datos del bit)
- Datos leídos del Bus (línea de dirección de 32 bits y 32 datos del bit)
- Datos escritos en el Bus (línea de dirección de 32 bits y 32 datos del bit)

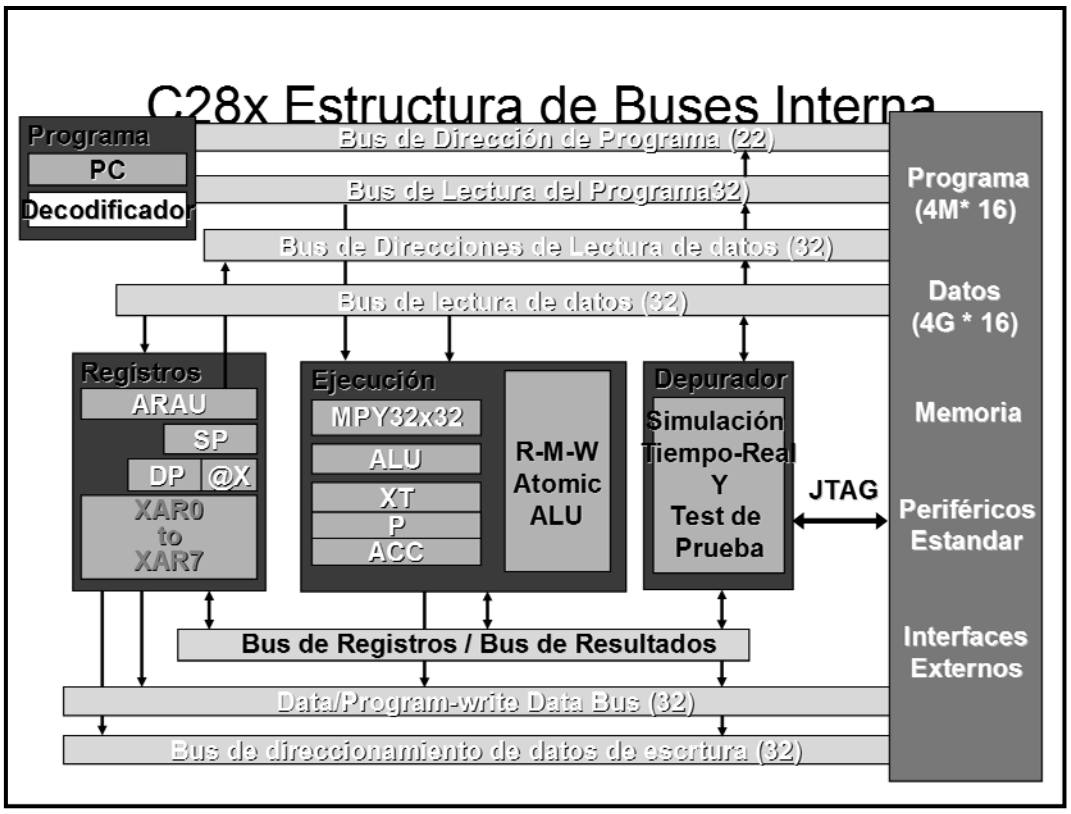


Figura 2.5: Estructura de buses del C28XX

Los Buses de datos de 32 bits permiten operaciones de 32 bits de un solo ciclo. Esta arquitectura múltiple del Bus, conocida como arquitectura del Bus Harvard permite al C28x traer una instrucción leer y escribir el valor de los datos en un solo ciclo. Todos los periféricos y memorias se unen a la memoria el Bus y se dará la prioridad a accesos de memoria.

1.2.7.- Unidad Aritmético Lógica Atómica

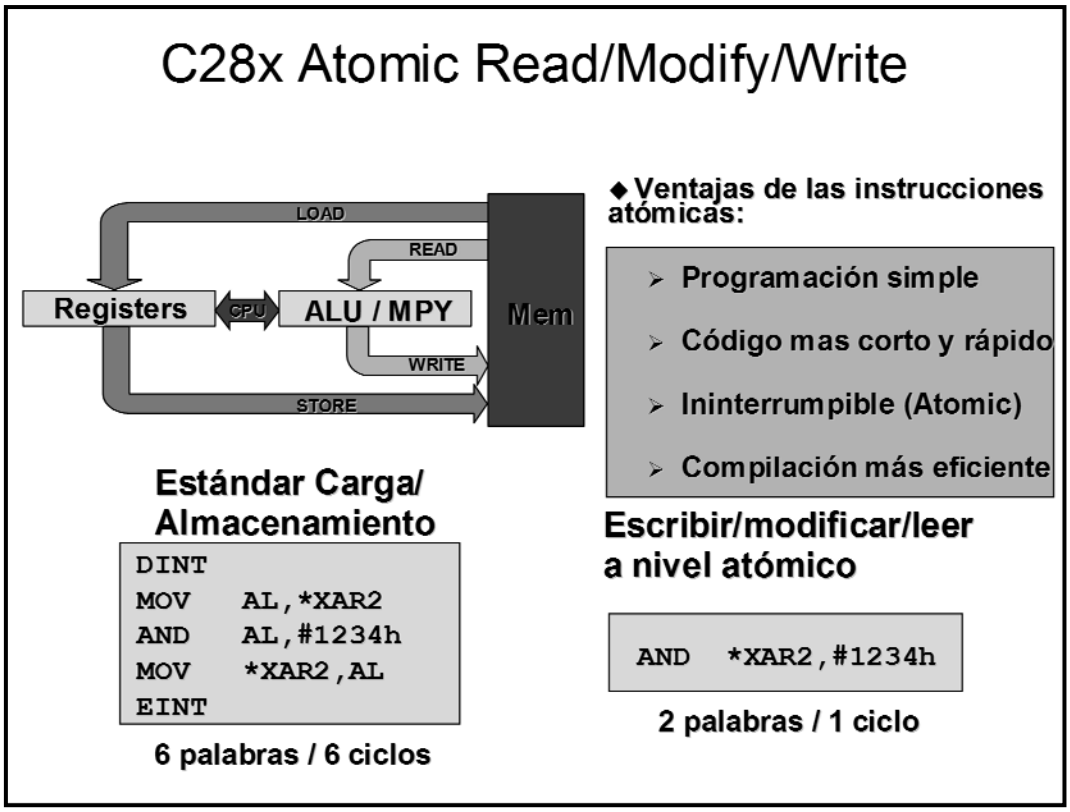


Figura 2.6: Instrucciones atómicas

Instrucciones atómicas (Figura 2.6) son las pequeñas instrucciones comunes que son ininterrumpibles. La capacidad atómica de ALU apoya instrucciones y el código que maneja tareas y procesos. Estas instrucciones se ejecutan generalmente varios ciclos más rápidas que en el código tradicional.

1.2.8.- Instrucción Pipeline

El F2812 utiliza un pipeline de 8 estados para maximizar el rendimiento de procesamiento (Figura 2.7). Este pipeline previene de escribir y leer de la misma localización a la vez que también permite al C28x ejecutar código a altas velocidades sin tener que recurrir a memorias rápidas.

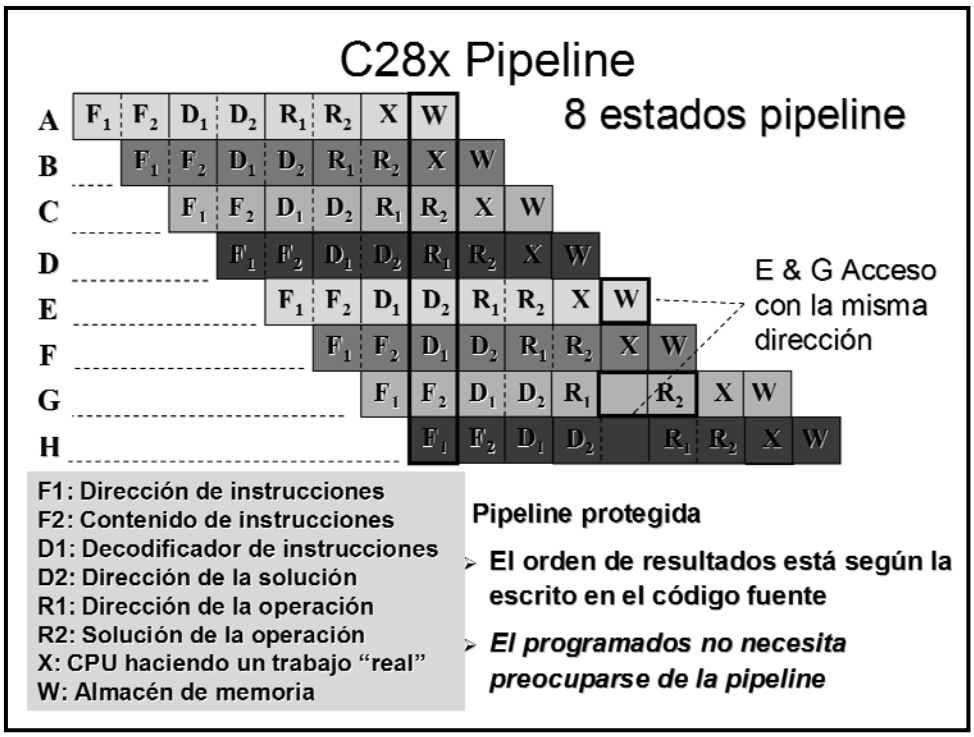


Figura 2.7: Instrucción Pipeline

Cada instrucción pasa a través de 8 etapas hasta la aplicación final. Una vez que el pipeline se llene de instrucciones, una instrucción se ejecuta por ciclo de reloj. Para un dispositivo 150MHz, esto se traduce a 6.67ns por la instrucción. Las etapas son:

- F1: Genera la dirección de instrucción en las líneas de la dirección del Bus del programa.
- F2: Lee la instrucción de líneas de datos del Bus del programa.
- D1: Descifra la instrucción
- D2: Calcula la información de la dirección para el operando de la instrucción
- R1: Carga la dirección del operando a las líneas de la dirección del Bus de los datos y/o del programa
- R2: Lee el operando
- X: Ejecuta la instrucción
- W: Escribe detrás el resultado a la memoria de los datos

1.2.9.- Mapa de memoria

La memoria en el F2812 se divide en espacio del programa y de los datos (Figura 2.8). Hay varios y diversos tipos de memorias disponibles que se pueden utilizar o en el programa o en el espacio de los datos. Incluyen memoria flash de acceso RAM (SARAM), SARAM ampliado, y la ROM que es programada con las rutinas del software del cargador o las tablas de estándar usadas en algoritmos relacionados con las matemáticas. El tamaño de palabra es de 16 bits.

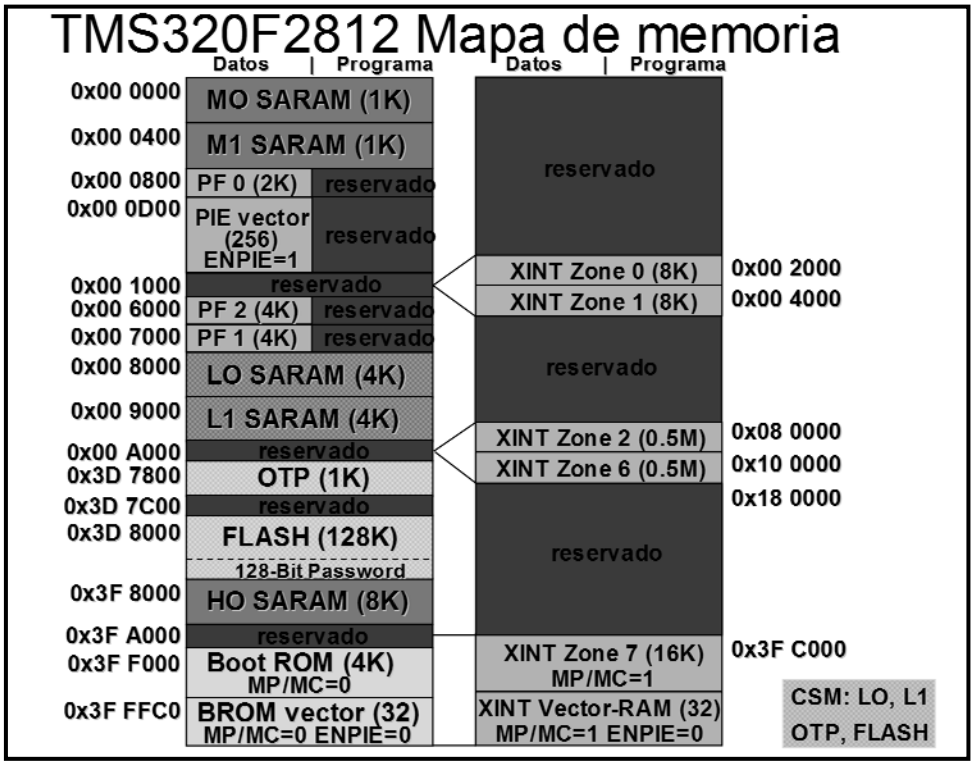


***Figura 2.8*:** Mapa de memoria del F2812

El F2812 puede acceder a ambas memorias por intervalos. El F2812 usa 32 bits de direcciones de datos y 22 bits para las direcciones del programa Esto permite un alcance total de la dirección de las palabras 4G (1 palabra = 16 bits) en el espacio de los datos y de 4M en el espacio del programa. Los bloques de la memoria en todos los diseños F2812 son mapeados uniformemente al programa y al espacio de los datos.

La memoria interna permanente consiste en un grupo de secciones de la memoria Flash, de un cargador ROM para hasta seis opciones del reset-arranque y de un área de tiempo programable (OTP). El FLASH y OTP se utilizan generalmente para almacenar el código de control para el uso de los datos que deben estar presentes en el reset.

Para cargar la información en las memorias del FLASH y de OTP hay que utilizar un programa dedicado a la transferencia directa, ése es también parte del ambiente integrado estudio del diseño del Code Composer Studio de Texas Instruments.

La memoria volátil está partida en 5 áreas (M0, M1, L0, L1 y H0) que se puedan utilizar como memoria del código y memoria de los datos.

PF0, PF1 y PF2 son los periféricos que cubren los registros del control y del estado de todas las unidades periféricas (la "memoria mapeada de los registros").

1.2.10.- Módulo del código de seguridad

Hay un módulo interno de seguridad disponible en todos los miembros de la familia F28x (Figura 2.9). Se basa en una contraseña 128-bit que sea escrita por el revelador del software en las 8 memorias pasadas del FLASH interno (0x3F 7FF8 a 0x3F 7FFF). Una vez que un patrón se escriba en esta área, todos los otros accesos a cualquiera de las áreas de memoria cubiertas por este módulo de la seguridad del código (CSM) se niegan, mientras el usuario no escribe un patrón idéntico en los registros de la contraseña del bastidor PF0.

NOTA: ¡Si se escribe cualquier patrón en el área de la contraseña por accidente, no hay manera de conseguir el acceso a este dispositivo nunca más! ¡Tenga mucho cuidado y no trastorne al técnico del laboratorio!

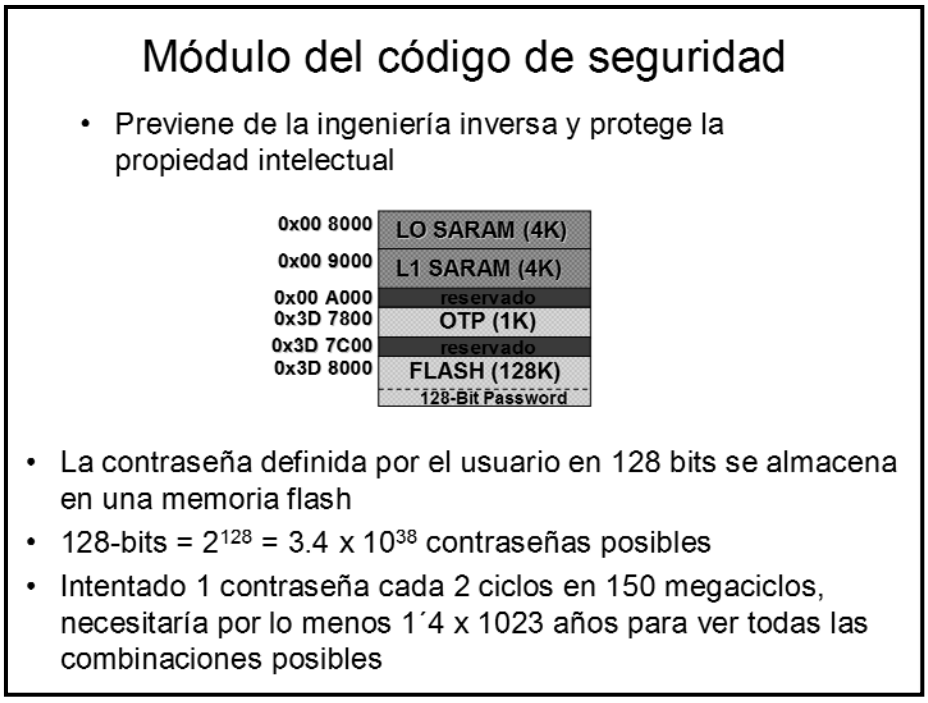


Figura 2.9: Código de seguridad.

1.2.11.- Respuesta a una interrupción

La respuesta rápida a una interrupción (Figura 2.10), con “grabación automática del contexto” almacena los registros críticos, dando por resultado un dispositivo que es capaz de mantener muchos acontecimientos asíncronos con estado latente mínimo. Aquí el "contexto" significa todos los registros que se necesitan ahorrar de modo que se pueda salir y realizar uno u otro proceso, después se vuelve exactamente donde se dejó. Esta característica ayuda a reducir los overheads de la rutina del servicio de la interrupción.

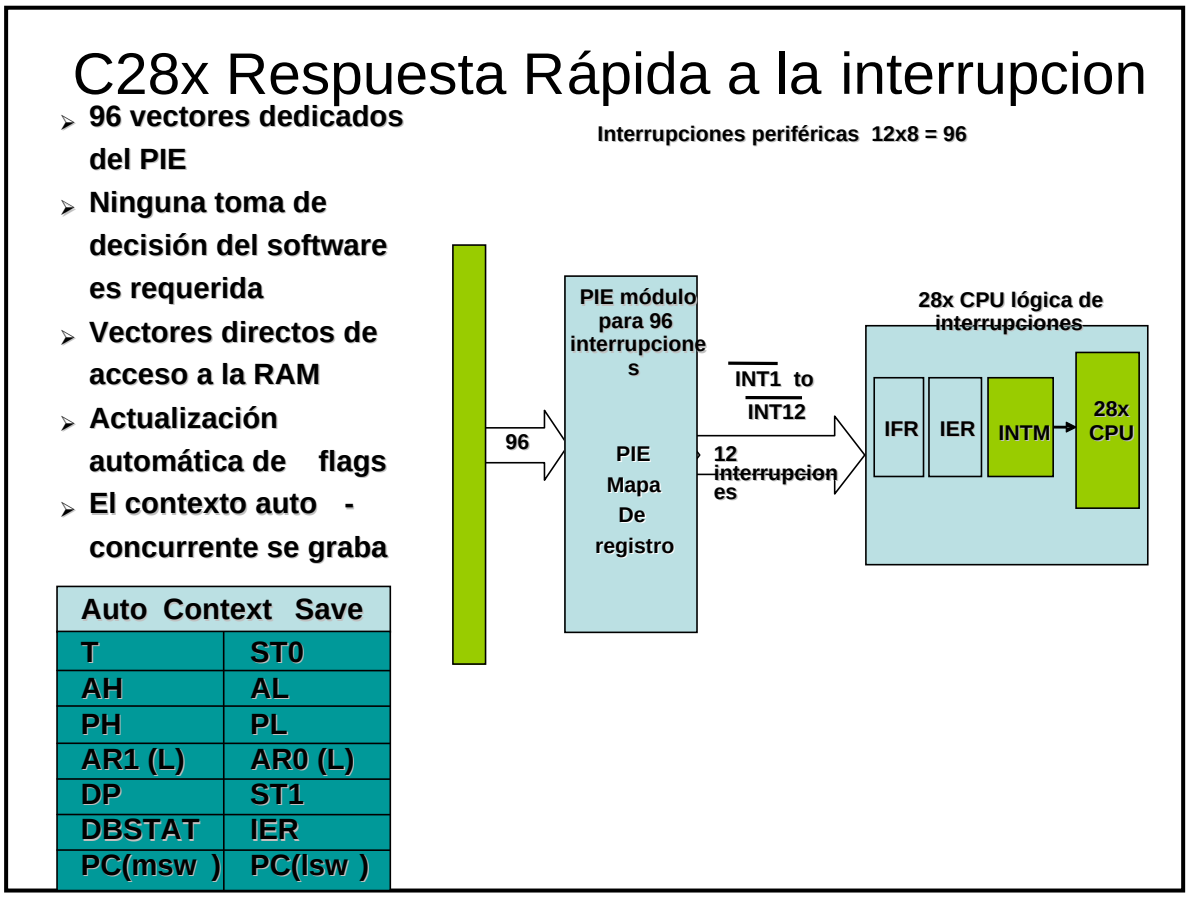


Figura 2.10: Interrupciones del C28XX

Se estudiará mas detalladamente en el sistema de la interrupciones de F2812's en el capítulo 7. El periférico de extensión de interrupciones (PIE) permite que el usuario especifique las rutinas individuales de servicio de interrupciones, hasta 96 acontecimientos externos e internos de la interrupción.

El auto contexto almacena 14 registros importantes de la CPU, como se muestra en la figura superior, en una memoria realizada a modo de pila, que es señalada por un registro del puntero de pila (SP). El apilado es parte de la memoria de los datos y debe residir en las palabras más bajas de 64K de la memoria de los datos.

1.2.12.- Modos de Funcionamiento

Al ser el F2812 uno de los miembros de las generaciones de punto fijo de los procesadores de señal numérica (DSP) en la familia TMS320C2000, tanto su código fuente como su código objeto son compatibles con el C27x (Tabla 2.11). Además, en el F2812 su código de fuente es compatible con el 24x/240x DSP y el código previamente escrito se puede volver a montar para funcionar en un dispositivo F2812. Esto permite la migración del código existente sobre el F2812.

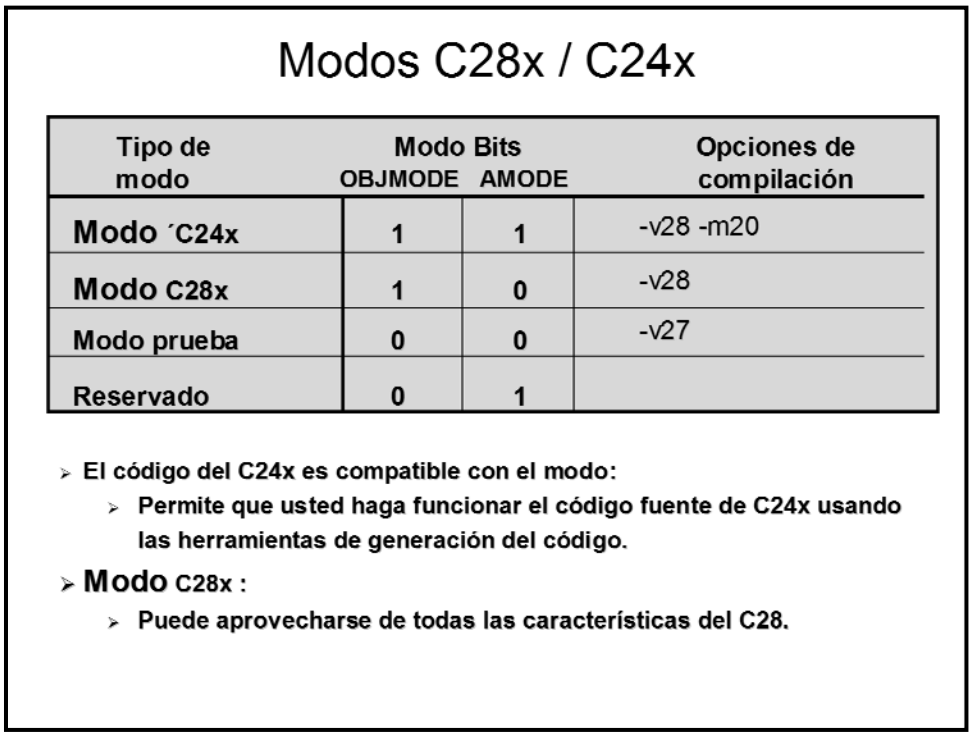


Tabla 2.11: Versiones compatibles.

El silicio del F28x puede funcionar en tres diversos modos:

- El modo C28x - se aprovecha de todas las características 32-bit del dispositivo
- El modo C24x - compatibilidad del código de fuente a los miembros 16-bit de la familia
- El modo C 27x - modo de funcionamiento intermedio, propósitos de prueba solamente.

Después del Reset, el dispositivo se comporta como un dispositivo de C27x. Para aprovecharse de la potencia completa de un dispositivo de C28x, el "OBJMODE" del control se debe fijar a 1. Si se está utilizando un programa generado por Ccompiler, la función del comienzo del ambiente de C toma por defecto fijar OBJMODE 1. Pero, si se desarrolla una solución en lenguaje ensamblador, la primera tarea después del reset será llevar al dispositivo a modo manual.

1.2.13.- Función Reset

Después de que se ejecute un reset válido al F2812 la secuencia siguiente depende de algunos pines externos a este DSC (Figura 2.11).

Si el pin "XMPNMC" está conectado a ‘1', el F2812 intenta cargar la instrucción siguiente de la dirección 0x3F FFC0 de la memoria externa en esta posición. Éste es el modo de "microprocesador", cargando instrucciones de memoria externa del código.

Si el pin "XMPNMC" está conectado a ‘0', el F2812 salta directamente en la memoria interna a la dirección 0x3F FFC0. Se llama a este modo de funcionamiento “Microcontrolador". El código en esta posición de memoria ha sido desarrollado por el fabricante para poder distinguir entre 6 diversas opciones de arranque para el F2812. La opción real es seleccionada por el estado de 4 pines más durante esta fase.

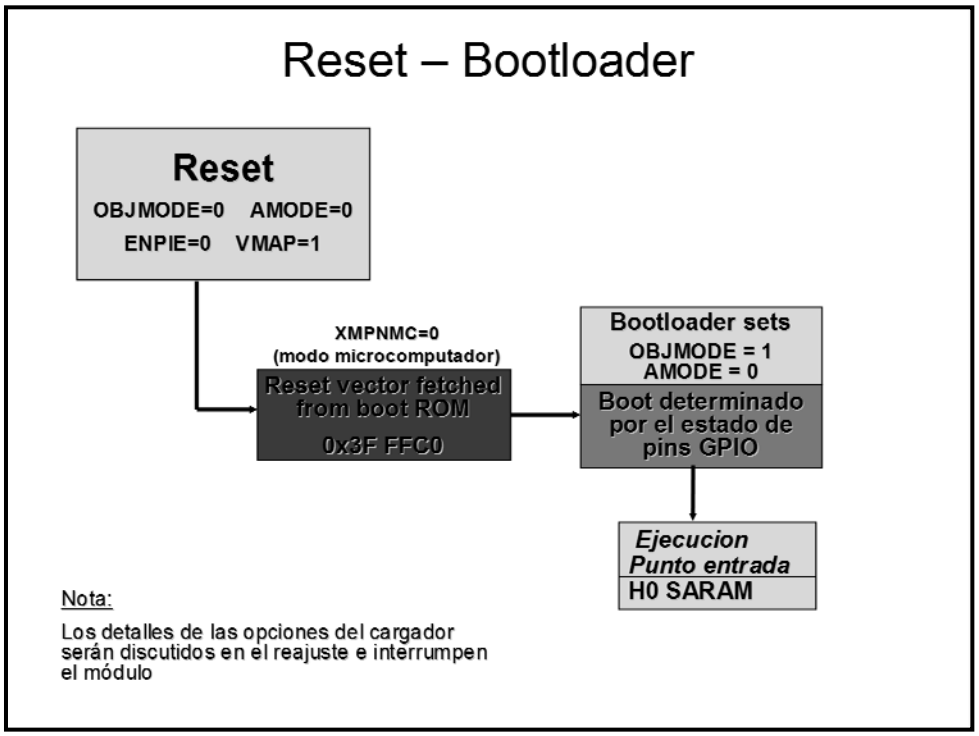


Figura 2.11: Reset del DSP

1.2.14.- Resumen de la arquitectura TMS320F2812

- Alto rendimiento DSP 32-bit
- 32 x 32 bits o doble MAC de 16 x 16 bits
- En nivel atómico leer-modificar-escribir las instrucciones
- Pipeline completamente protegido de 8 etapas
- Encargado de la respuesta rápida de interrupción
- Memoria flash en chip de 128Kw
- Módulo de la seguridad (CSM)
- Dos event manager
- Módulo de 12-bit ADC
- 56 pines compartidos de GPIO
- Contador de tiempo perro guardián
- Periférico de comunicaciones

1.3.- PLACA EZDSP DE TMS320F2812

1.3.1.- Introducción

En este tema se describen las operaciones a nivel de placa del eZdspTM F2812 del Procesador Digital de la Señal TMS320F2812 de Texas Instruments.

El eZdspTM F2812 (Figura 3.1) es un módulo independiente que permite a los ingenieros y a los programadores la evaluación de las características del DSP TMS320F2812 para determinar la aplicabilidad del procesador a los diferentes requerimientos del sistema. Los evaluadores pueden crear un software para ejecutar en la placa o expandir el sistema en una gran variedad de posibilidades.

Cuando nombremos el eZdsp nos referiremos al eZdspTM F2812. Este capítulo tiene una descripción del eZdsp del Procesador Digital de la Señal TMS320F2812, sus características dominantes y el diagrama de bloques de la placa del circuito.

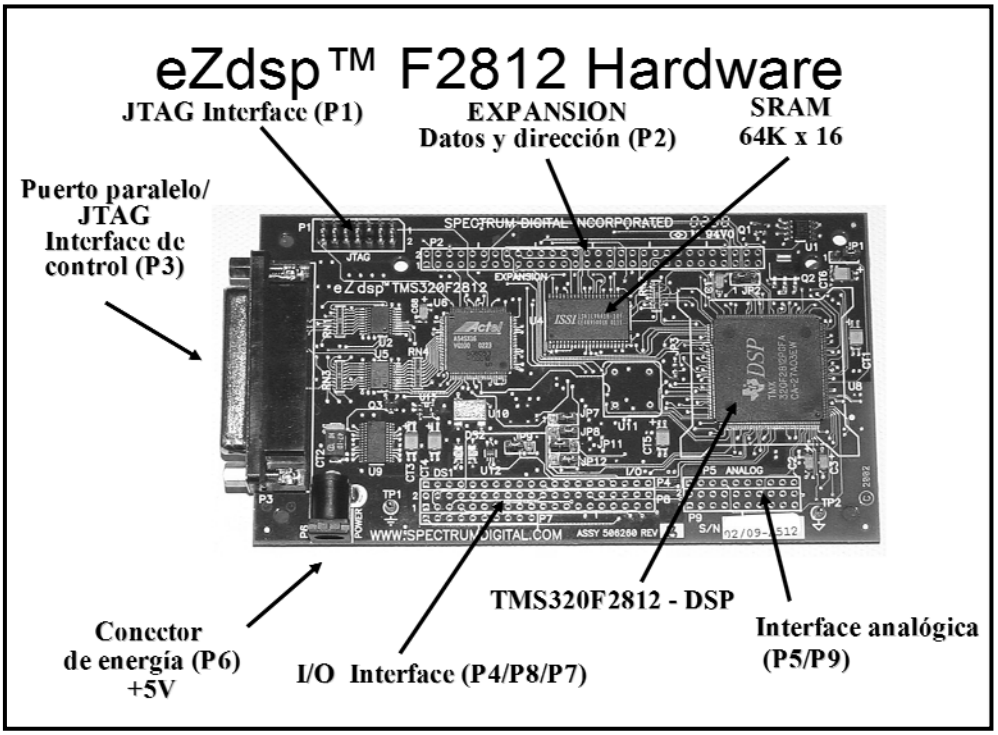


Figura 3.1: Placa eZdsp del F2812.

1.3.2.- Descripción del eZdspTM F2812

El eZdspTM F2812 es un módulo independiente que ayuda a los evaluadores a examinar el Procesador Digital de la Señal para determinar si es útil para las especificaciones de su aplicación. Además este módulo es una excelente plataforma para mejorar y ejecutar software del procesador TMS320F2812.

El eZdspTM F2812 incluye el TMS320F2812A DSP. El eZdspTM F2812 permite una rápida verificación del código de F2812A. Está provisto con dos conectores de expansión por si es necesaria alguna evaluación no incluida en la configuración.

Para simplificar el desarrollo y tener un tiempo menor de depurado, se incluye con el C2000 Tools Code Composer. Además está provisto con un conector JTAG que sirve como interfase para los emuladores, operando con otros depuradores para proveer un depurado lenguaje ensamblador y C de alto nivel.

1.3.3.- Características dominantes del eZdspTM F2812

El eZdspTM F2812 tiene los siguientes componentes:

- Procesador Digital de la Señal TMS320F2812A
- Velocidad de operación de 150 MIPS
- Un chip de RAM de 1k word
- Un chip de memoria Flash de 128k words
- Un chip de memoria SRAM de 64k words
- 30 MHz en placa. Un reloj desde la puerta de array de la CPU
- 2 Conectores de expansión (Analógica, E/S)
- Controlador JTAG IEEE 1149.1
- Operaciones solo de 5 voltios con el adaptador AC suministrado
- Driver TI F28xx Code Composer Studio tools
- Conector de emulación JTAG IEEE 1149.1

1.3.4.- Descripción funcional del eZdspTM F2812

La figura 3.2 muestra un diagrama de bloques de la configuración básica del **eZdspTM F2812**. Las interfases principales del eZdsp son las interfases JTAG y la interfaz de expansión.

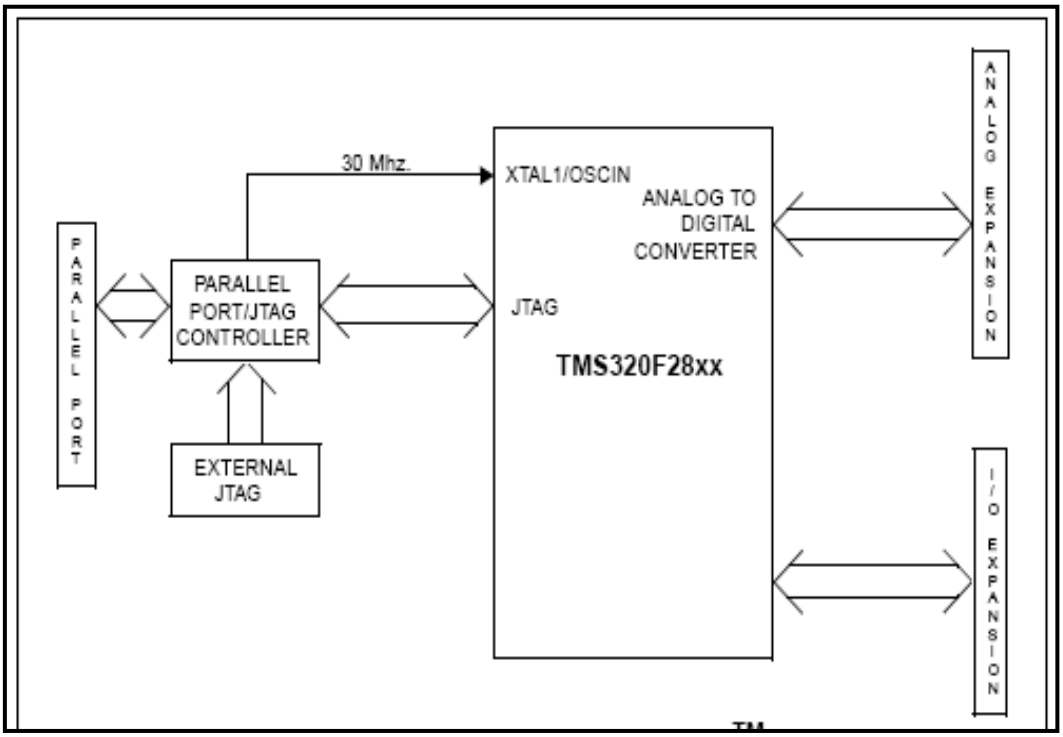


Figura 3.2: Configuración básica del eZdspTM F2812

1.3.5.- La placa eZdspTM F2812

El eZdspTM mide 5.25 x 3.0 pulgadas, es un circuito a doble cara (Figura 3.3), es decir, tiene pistas por las dos caras; está alimentado a 5 voltios externos de una fuente de alimentación.

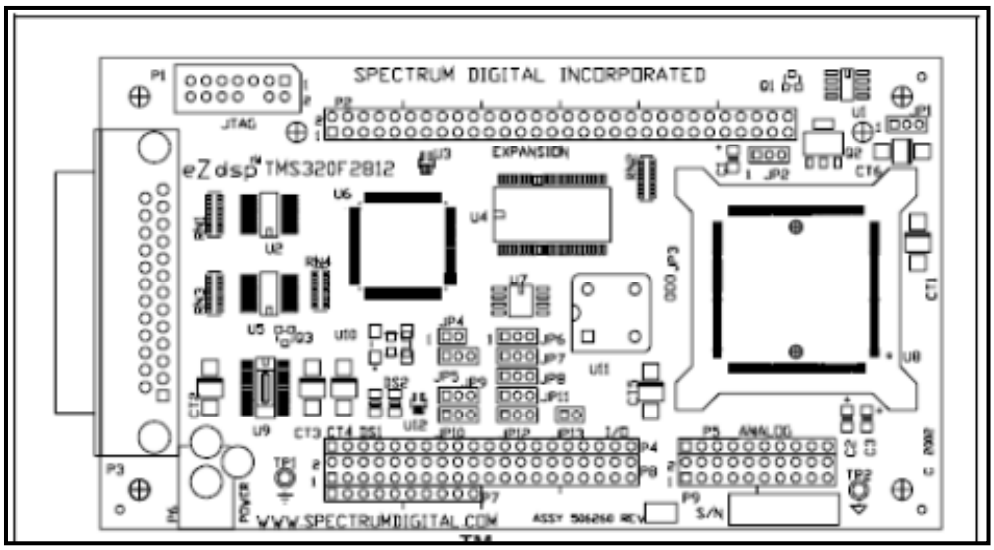


Figura 3.3: Circuito esquemático de la placa eZdsp F2812

1.3.6.- Conector de alimentación

El eZdspTM F2812 está alimentado a 5 voltios de una fuente externa, incluida en la unidad. La unidad requiere 500mA. La alimentación se le aplicará mediante el conector P6. Si la placa de expansión esta conectada al eZdsp se necesitará un amperaje mayor.

1.3.7.- La memoria del eZdspTM F2812

El eZdsp incluye las siguientes memorias en chips:

- Flash de 128k x 16
- RAM (SARAM) de 2 bloques de 4k x 16 accesos simples
- SARAM de 1 bloque de 8k x 16
- SARAM de 2 bloques de 1k x 16

Además hay un chip SRAM de 64k x 16. El procesador en el eZdsp puede ser configurado mediante modo de carga o el modo de no carga.

El eZdsp puede cargar su RAM para depurar o la FLASH ROM para ejecutar. Para proyectos largos de software es recomendable comenzar con un depurado inicial en un ambiente de RAM total. Con precaución a las E/S mapeadas en el software la aplicación puede ser muy fácil.

1.3.8.- Mapa de memoria

La tabla 3.4 muestra el mapa de memoria de configuración del eZdspTM F2812.

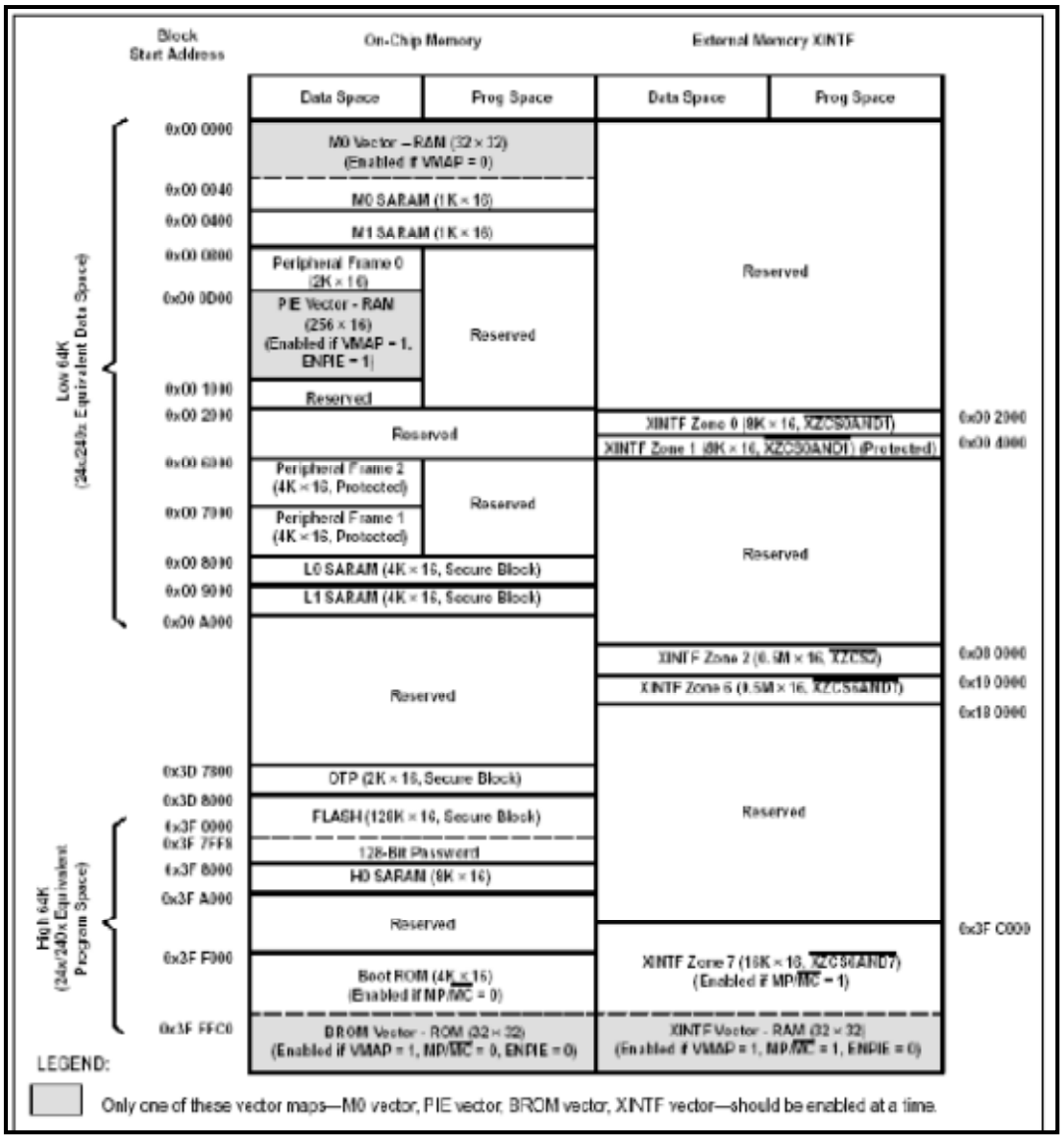


Tabla 3.4: Distribución del mapa de memoria.

Nota: El chip de la memoria flash tiene un código de seguridad con una clave para prevenir visiblemente cuando está siendo usada.

1.3.9.- Conectores eZdspTM F2812

El **eZdspTM F2812** tiene cinco conectores. El pin 1 de cada conector es para ser identificado mediante un cuadrado de soldadura. La función de cada conector se muestra en la siguiente tabla:

Connector	Function
P1	JTAG Interface
P2	Expansion
P3	Parallel Port/JTAG Controller Interface
P4/P8/P7	I/O Interface
P5/P9	Analog Interface
P6	Power Connector

Tabla 3.5: Funciones de cada conector

La figura siguiente muestra la posición de cada conector en la placa eZdsp.

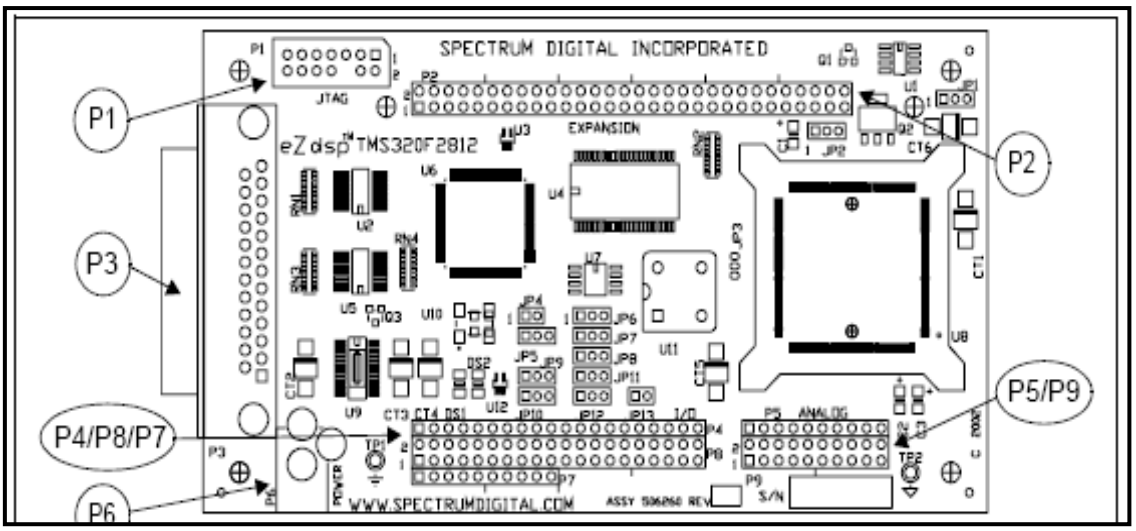


Figura 3.6: Distribución de los conectores en la placa

1.3.10.- Interfaz P1, JTAG

El eZdspTM F2812 está suministrado con una interfaz de 14 pins de cabecera, P1. Este es el interfaz estándar usado para las emulaciones del JTAG por Texas Instruments para los DSPs.

La posición de los 14 pins en el conector P1 está mostrada a continuación.

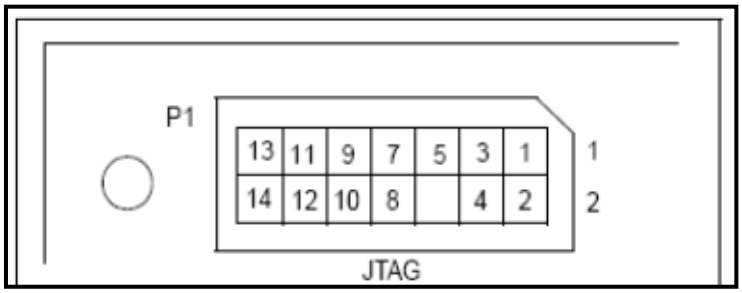


Figura 3.7: Interfaz P1

La definición de P1, que tiene las señales JTAG se muestra en la siguiente tabla:

Pin #	Signal	Pin #	Signal
1	TMS	2	TRST-
3	TDI	4	GND
5	PD (+5V)	6	**no pin**
7	TDO	8	GND
9	TCK-RET	10	GND
11	TCK	12	GND
13	EMU0	14	EMU1

Tabla 3.8: Definición de pines del P1

1.3.11.- Interfaz De Expansión P2

Las posiciones de los 60 pins del conector P2 están mostradas en la siguiente tabla visto desde el final del eZdsp.

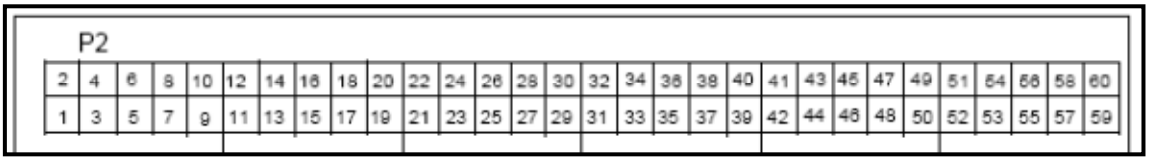


Tabla 3.9: Posicionamiento de los pines del P2

La definición de P2, que tiene la interfaz de las señales de E/S se muestra a continuación.

Pin #	Signal	Pin #	Signal
1	+5V	2	+5V
3	XD0	4	XD1
5	XD2	6	XD3
7	XD4	8	XD5
9	XD6	10	XD7
11	XD8	12	XD9
13	XD10	14	XD11
15	XD12	16	XD13
17	XD14	18	XD15
19	XA0	20	XA1
21	XA2	22	XA3
23	XA4	24	XA5
25	XA6	26	XA7
27	XA8	28	XA9
29	XA10	30	XA11
31	XA12	32	XA13
33	XA14	34	XA15
35	GND	36	GND
37	XZCS0AND1n/PSn	38	XZCS2n/DSn
39	XREADY	40	ISn
41	XRnW	42	STRBn
43	XWE	44	XRDn
45	+3.3V/BR-	46	XNMN/INT3
47	XRSn/RSn	48	
49	GND	50	GND
51	GND	52	GND
53	XA16	54	XA17
55	XA18	56	XHOLDn
57	XHOLDAn	58	
59		60	

Tabla 3.10: Definición de pines del P2

1.3.12.- Interfaz del puerto paralelo/JTAG, P3

El eZdspTM F2812 usa un dispositivo normal de interfaz para el puerto paralelo – JTAG. Este dispositivo incorpora una interfaz de puerto paralelo que tiene ECP, EPP y SPP8/Comunicaciones bidireccionales. Este dispositivo tiene acceso directo a la interfaz integrada del JTAG. Los drivers del C2000 Code Composer están equipados en los módulos de eZdsp.

1.3.13.- Interfaz E/S, P4/P8/P7

Los conectores P4, P8 y P7 presentan las señales de E/S desde el DSP. El layout de estos conectores es mostrado a continuación:

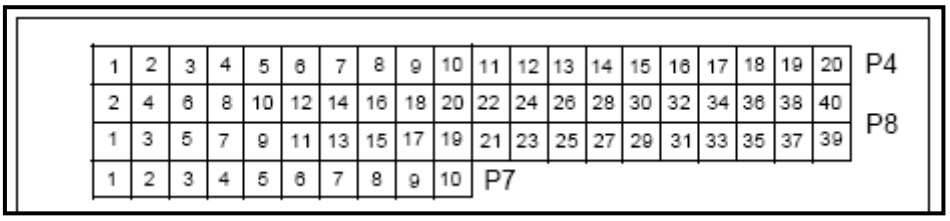


Figura 3.11: Distribución de pines del P4, P7 y P8

La placa realizada para la interconexión entre el dsp y el osciloscopio (Anexo 1) va conectada en el grupo de pines P8 mediante un cable plano

La descripción de los conectores P4/P8 está mostrada a continuación

P4 Pin #	P4 Signal	P8 Pin #	P8 Signal	P8 Pin #	P8 Signal
1	+5 Volts	1	+5 Volts	2	+5 Volts
2	XINT2/ADCSOC	3	SCITXDA	4	SCIRXDA
3	MCLKXA	5	XINT1n/XBIOn	6	CAP1/QEP1
4	MCLKRA	7	CAP2/QEP2	8	CAP3/QEP11
5	MFSXA	9	PWM1	10	PWM2
6	MFSRA	11	PWM3	12	PWM4
7	MDXA	13	PWM5	14	PWM6
8	MDRA	15	T1PWM/T1CMP	16	T2PWM/T2CMP
9		17	TDIRA	18	TCLKINA
10	GND	19	GND	20	GND
11	CAP5/QEP4	21		22	XINT1N/XBIOn
12	CAP6/QEP12	23	SPISOMA	24	SPIOMA
13	T3PWM/T3CMP	25	SPICLKA	26	SPISTEA
14	T4PWM/T4CMP	27	CANTXA	28	CANRXA
15	TDIRB	29	XCLKOUT	30	PWM7
16	TCLKINB	31	PWM8	32	PWM9
17	XF/XPLLDISn	33	PWM10	34	PWM11
18	SCITXDB	35	PWM12	36	CAP4/QEP3
19	SCIRXDB	37	T1CTRIP/PDPINT	38	T3CTRIP/PDPINTBn
20	GND	39	GND	40	GND

Tabla 3.12: Definición de pines de P4 y P8

La descripción del conector P7 se muestra en la tabla que sigue:

P7 Pin #	P7 Signal
1	C1TRIPn
2	C2TRIPn
3	C3TRIPn
4	T2CTRIPn/EVASOCn
5	C4TRIPn
6	C5TRIPn
7	C6TRIPn
8	T4REPn/EVBSOCn
9	
10	GND

Tabla 3.13: Definición de pines del P7

1.3.14.- Interfaz analógica P5/P9

La posición de los 30 pins en los conectores P5/P9 se muestra en la figura siguiente, visto desde arriba del eZdsp.

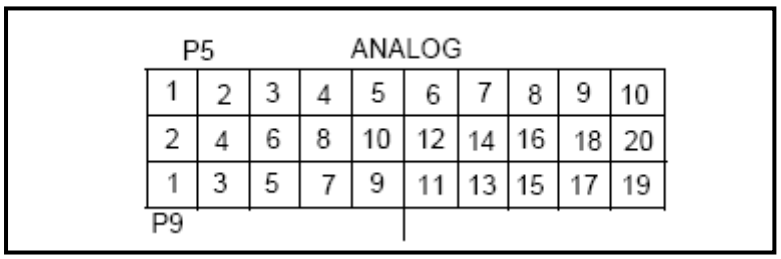


Figura 3.14: Interfaz P5 y P9

La descripción de las señales P5/P9 está mostrada en la tabla continua.

P5 Pin #	Signal	P9 Pin #	Signal	P9 Pin #	Signal
1	ADCINB0	2	GND	2	ADCINA0
2	ADCINB1	4	GND	4	ADCINA1
3	ADCINB2	6	GND	6	ADCINA2
4	ADCINB3	8	GND	8	ADCINA3
5	ADCINB4	10	GND	10	ADCINA4
6	ADCINB5	12	GND	12	ADCINA5
7	ADCINB6	14	GND	14	ADCINA6
8	ADCINB7	16	GND	16	ADCINA7
9	ADCREFM	18	GND	18	VREFLO
10	ADCREFP	20	GND	20	

Tabla 3.15: Definición de pines de P5 y P9

1.3.15.- Conector de alimentación P6

La alimentación (5 voltios) se introduce dentro del eZdspTM F2812 mediante el conector P6. El conector tiene un diámetro exterior de 5.5 mm. Y un diámetro interior de 2 mm. La posición del conector P6 se muestra a continuación.

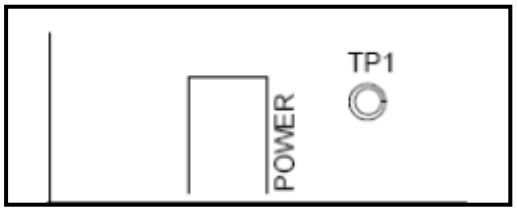


Figura 3.16: Conector P6 de alimentación.

El diagrama de P6, que tiene la alimentación es mostrado abajo.

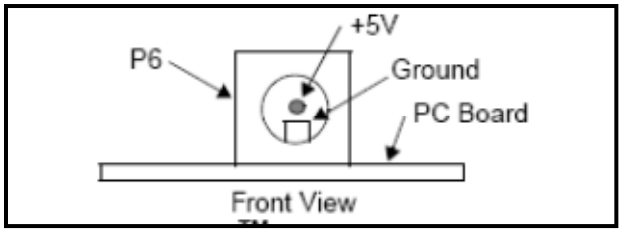


Figura 3.17: Esquema de alimentación de P6

1.3.16.- Números de parte del conector

La siguiente tabla muestra los números de los conectores que pueden ser usados en el eZdspTM F2812.

Connector	Male Part Numbers	Female Part Numbers
P1	SAMTEC TSW-1-10-07-G-T	SAMTEC SSW-1-10-01-G-T
P2	SAMTEC TSW-1-20-07-G-T	SAMTEC SSW-1-20-01-G-T

*SSW or SSQ Series can be used

Tabla 3.18: Conectores P1 y P2

1.3.17.- Puentes eZdspTM F2812

El **eZdspTM F2812** tiene 9 puentes posibles para el uso que determinan como deben ser usadas en el eZdspTM F2812. La tabla 3.19 es una lista de los puentes y su función. Las siguientes secciones describen el uso para cada puente.

Jumper #	Size	Function	Position As Shipped From Factory
JP1	1 x 3	XMP/MCn	2-3
JP2	1 x 3	Flash Power Supply	1-2
JP6	1 x 3	Test Mode Select	2-3
JP7	1 x 2	Boot Mode 3	2-3
JP8	1 x 3	Boot Mode 2	2-3
JP9	1 x 3	PLL Disable	1-2
JP10	1 x 3	Connect XF to LED DS2	1-2
JP11	1 x 3	Boot Mode 1	1-2
JP12	1 x 3	Boot Mode 0	2-3

Tabla 3.19: Posibles puentes en la placa eZdsp

A menos que se indicare en forma diferente, todos los puentes 1x3 se deben instalar en la posición 1-2 o 2-3

En la figura 3.20 se muestran las posiciones de los puentes en el eZdspTM F2812.

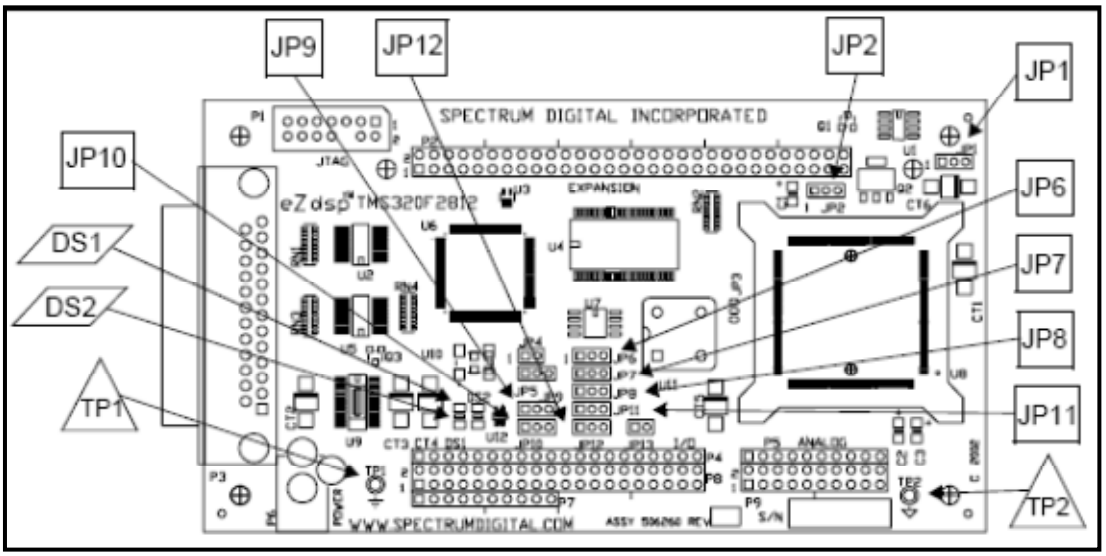


Figura 3.20: Diagrama de puentes en eZdsp.

1.3.18.- Selección XMP/MCn, JP1

El puente JP1 es usado para seleccionar la opción XMP/MCn. La selección 1-2 permite al DSP operar en el modo microcontrolador. La opción 2-3 permite al DSP operar en el modo Microprocesador. Las posiciones son mostradas a continuación.

Position	Function
1-2	Microcontroller mode
2-3 *	Microprocessor mode

* as shipped from factory

Tabla 3.21: Puente selección Microcontrolador o Microprocesador

1.3.19.- Fuente de alimentación flash, JP2

El puente JP2 se usa para administrar voltaje al DSP para la programación en chip de la memoria Flash. Este puente siente está en la posición 1-2. Esto se muestra en la siguiente tabla.

Position	Function
1-2 *	Programming voltage supplied to DSP
2-3	Programming voltage not supplied to DSP

* always in this position

Tabla 3.22: Puente selección voltaje al DSP

1.3.20.- Selección del modo test, JP6

El puente JP2 es usado para determinar si el eZdsp podrá operar en el modo test o en el modo user. La selección 1-2 pone al eZdsp en el modo test. Si la posición es 2-3 el eZdsp se pondrá en modo user. Normalmente siempre se usa la posición 2-3. Las posiciones se muestran a continuación.

Position	Function
1-2	Test mode
2-3 *	User Mode

* always use this position

Tabla 3.23: Puente para seleccionar el modo test.

1.3.21.- Selección del modo de carga, JP7, JP8, JP11, JP12

Los puentes JP7, JP8, JP11, JP12 se usan para determinar e que modo se usará la descarga de código en el DSP. Las opciones son las siguientes:

JP7, BOOT3 SCITXDA	JP8, BOOT2 MDXA	JP8, BOOT2 MDXA	JP8, BOOT2 MDXA	MODE
1	X	X	X	FLASH
0	1	X	X	SPI
0	0	1	1	SCI
0	0	1	0	H0 *
0	0	0	1	OTP
0	0	0	0	PARALLEL

Tabla 3.24: Puentes para seleccionar el tipo de descarga del código.

1.3.22.- Desactivación del PLL, JP9

El puente JP9 se usa para activar/desactivar el uso de Phase Lock Loop (PLL) lógico en el DSP. La selección de la posición 1-2 activa el uso de PLL. Si ponemos el puente en la posición 2-3 el PLL es desactivado. Las posiciones se ven reflejadas en la tabla 5.15.

Position	Function
1-2 *	PLL Enabled
2-3	PLL disabled

* as shipped from the factory

Tabla 3.25: Puente para activación/desactivación del PLL.

1.3.23.- Selección de conexión XF bit a LED DS2

El puente JP10 se usa para determinar si el bit XF desde el procesador es conectado al LED DS2. Si la posición es 1-2 el XF es conectado al LED DS2 a través del buffer. Si la posición es 2-3 la selección desconecta el bit XF de LED DS2. La tabla siguiente muestra las opciones.

Position	Function
1-2 *	XF bit connected to LED DS2
2-3	XF bit not connected to LED DS2

* as shipped from the factory

Tabla 3.26: Conexión del bit XF al led DS2.

1.3.24.- LEDs

El eZdspTM F2812 tiene dos diodos que emiten luz. DS1 indica la presencia de los 5 voltios y normalmente está encendido cuando la placa está alimentada. DS2 está bajo el software y esta unido al pin XF a través del buffer del DSP (Tabla 3.27).

LED #	Color	Controlling Signal
DS1	Green	+5 Volts
DS2	Green	XF bit (XF high = on)

Tabla 3.27: Iluminación de leds si la alimentación esta conectada.

1.3.25.- Puntos de prueba

El eZdspTM F2812 tiene dos puntos de prueba, como se observa en la tabla 5.28.

Test Point	Signal
TP1	Ground
TP2	Analog Ground

Tabla 3.28: Puntos de prueba.

1.3.26.- Disposición del Hardware del laboratorio

Las siguientes figuras muestran el hardware que se utiliza en el laboratorio. La base es el TMS320F2812 DSP 32-bit en el eZdspF2812 de Spectrum Digital´s. Todos los periféricos internos están disponibles a través de los conectores. En el JTAG el emulador se conecta con el PC usando un cable paralelo como el de la impresora.

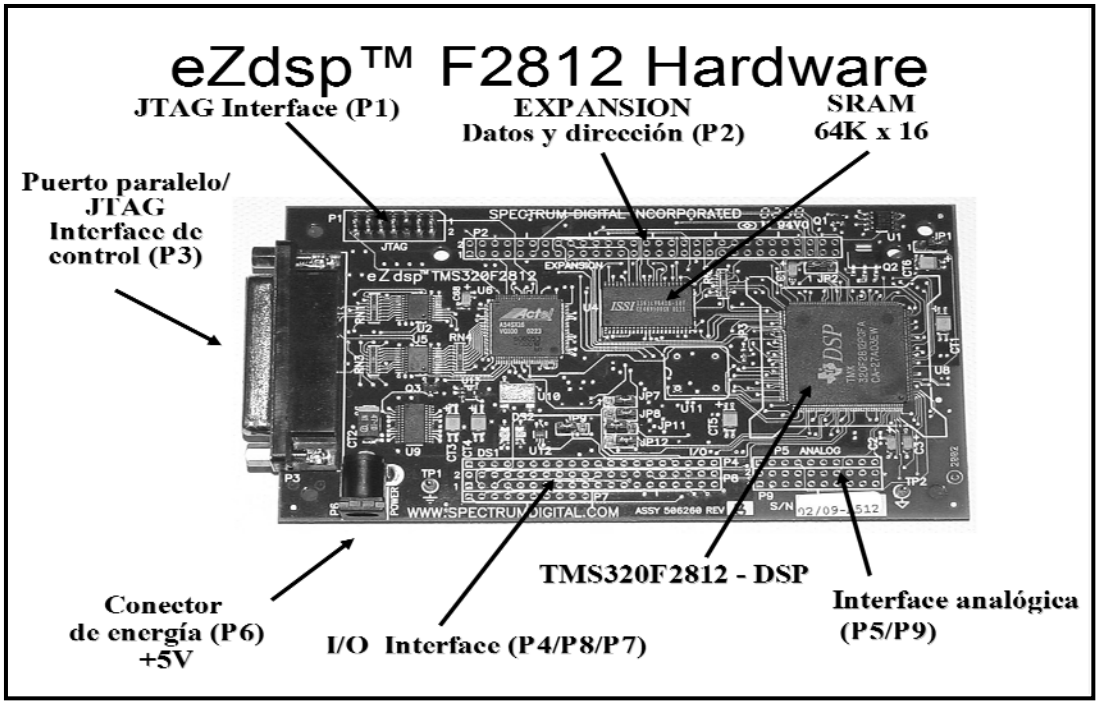


Figura 3.29: Disposición del hardware en la placa eZdsp F2812

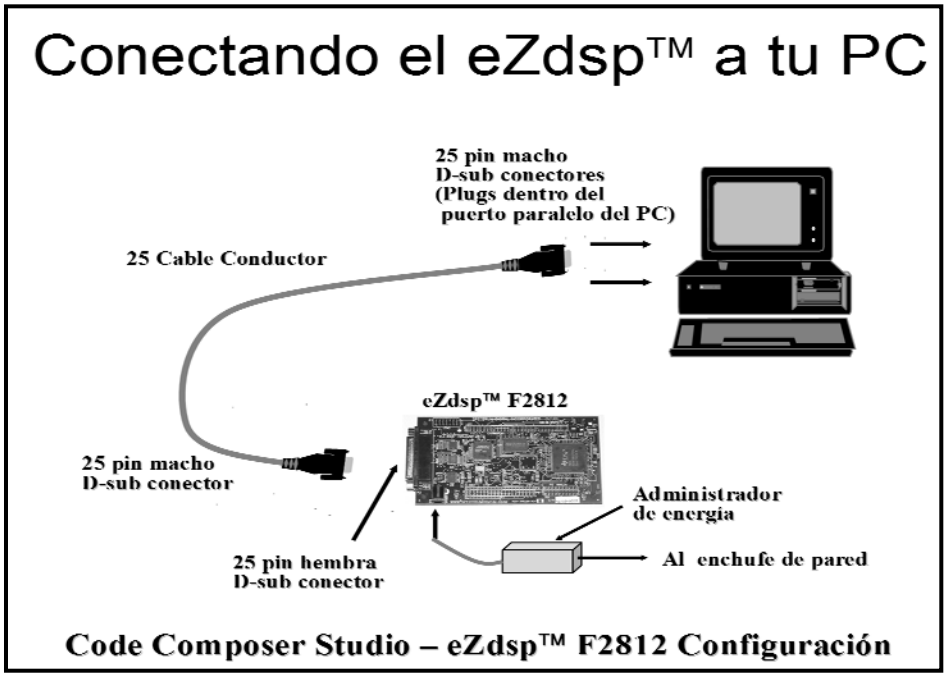


Figura 3.30: Conexión de la placa eZdsp a un PC.

1.4.- ENTRADAS/SALIDAS DIGITALES (Digital I/O)

1.4.1.- Introducción

En este capítulo se introducen las entradas y salidas digitales que permiten el intercambio de información con el exterior mediante señales de dos niveles, se vera el diagrama de bloques, la estructura que tienen los diferentes puertos así como sus registros específicos.

También se dará un repaso al reloj del sistema y al perro guardián. De este ultimo veremos su uso y la forma de desactivarlo.

1.4.2- Diagrama de bloques del F2812

Todas los periféricos del C28x están mapeados en la memoria de de datos. En la figura 4.1 se incluye una descripción gráfica de la estructura interna.

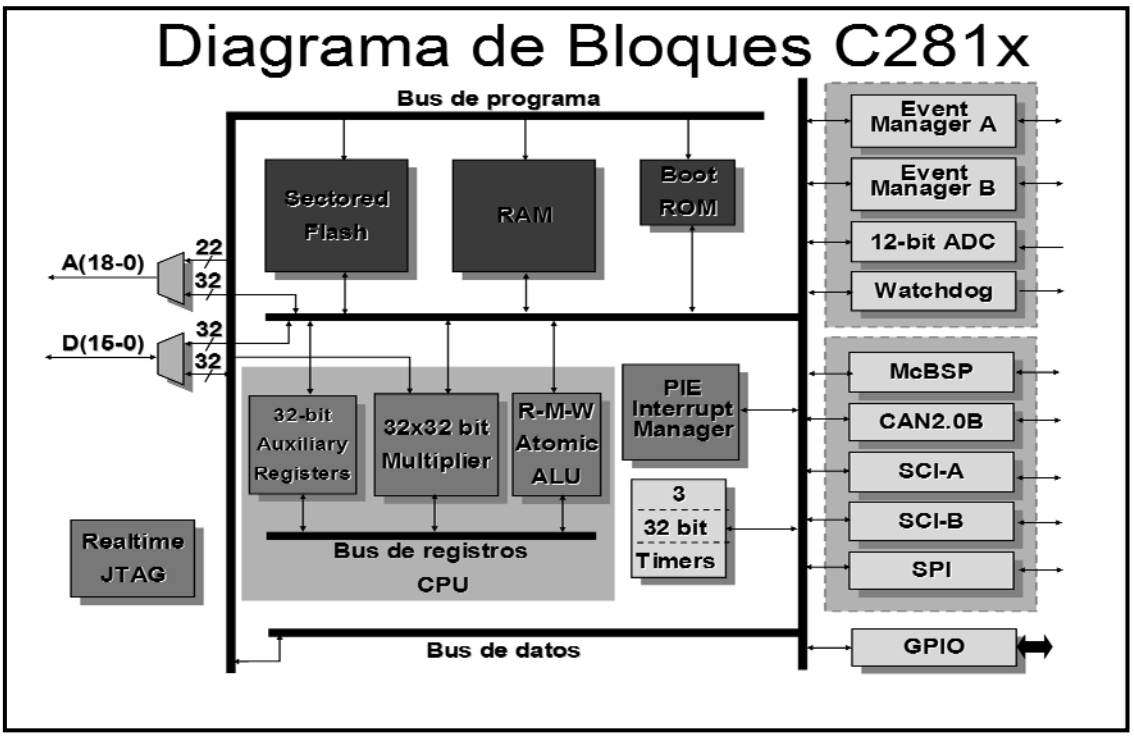


Figura 4.1: Diagrama de bloques del C281X.

1.4.3.- Los Periféricos

Todos los registros periféricos están agrupados alrededor de los "Peripheral Frames", PF0, PF1 y PF2. Estos "frames" son zonas de memoria donde están mapeados los registros.. La zona PF0 incluye registros para controlar la velocidad interna de la memoria FLASH, así como el acceso que mide el tiempo a la SARAM interna (SARAM es abreviatura de "Single Access RAM", que significa que se puede hacer un acceso en un ciclo de reloj). La memoria FLASH es la memoria interna no volátil, normalmente usada para el código del programa y para datos que deben estar presentes a lo largo del tiempo.

La zona PF1 contiene muchos de los registros de control de los periféricos.

La zona PF2 está reservada para el bloque de registros CAN "Controller Área Network" es una conexión estable muy extensamente utilizada en vehículos para comunicar todo con el sistema electrónico y poder controlarlo.

Ambas zonas se encuentran en la parte centrar izquierda del mapa de memoria de la figura 4.2.

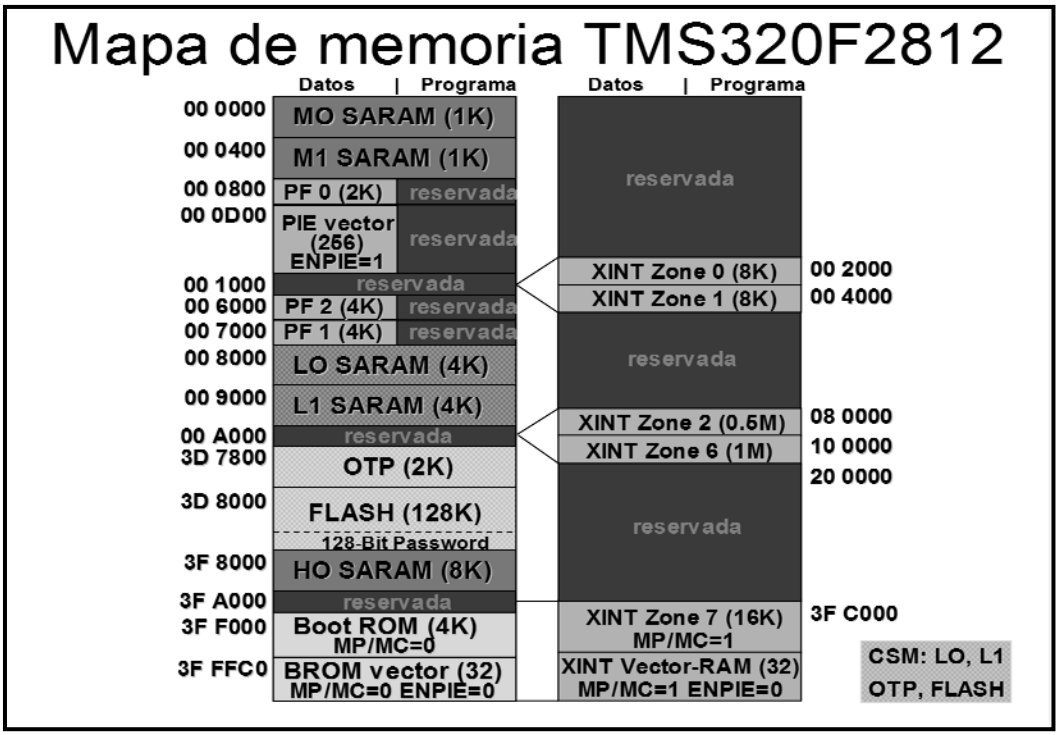


Figura 4.2: Mapa de memoria del TMS320F2812

Varias zonas de la memoria están protegidas mediante contraseñas por el "Code Security Module". Esto se utiliza para prevenir la ingeniería inversa. Una vez que se programe el área de la contraseña, cualquier acceso a las áreas aseguradas se concede solamente cuando la contraseña correcta se incorpora en un área especial de PF0.

1.4.4.- Unidad E/S Digital

Todas las entradas/salidas (I/O) digitales están agrupadas en puertos, llamados GPIO-A, B, D, E, F y G. GPIO significa “general purpose input ouput” (Figura 4.3). El C28x está equipado con muchas unidades internas, no todas las características se podrían conectar en los pins dedicados del dispositivo en cualquier momento. La solución es multiplexar. Esto es, un único pin físico del dispositivo puede ser usado por 2 (a veces 3) funciones diferentes y esto le permite al programador decidir que función es seleccionada.

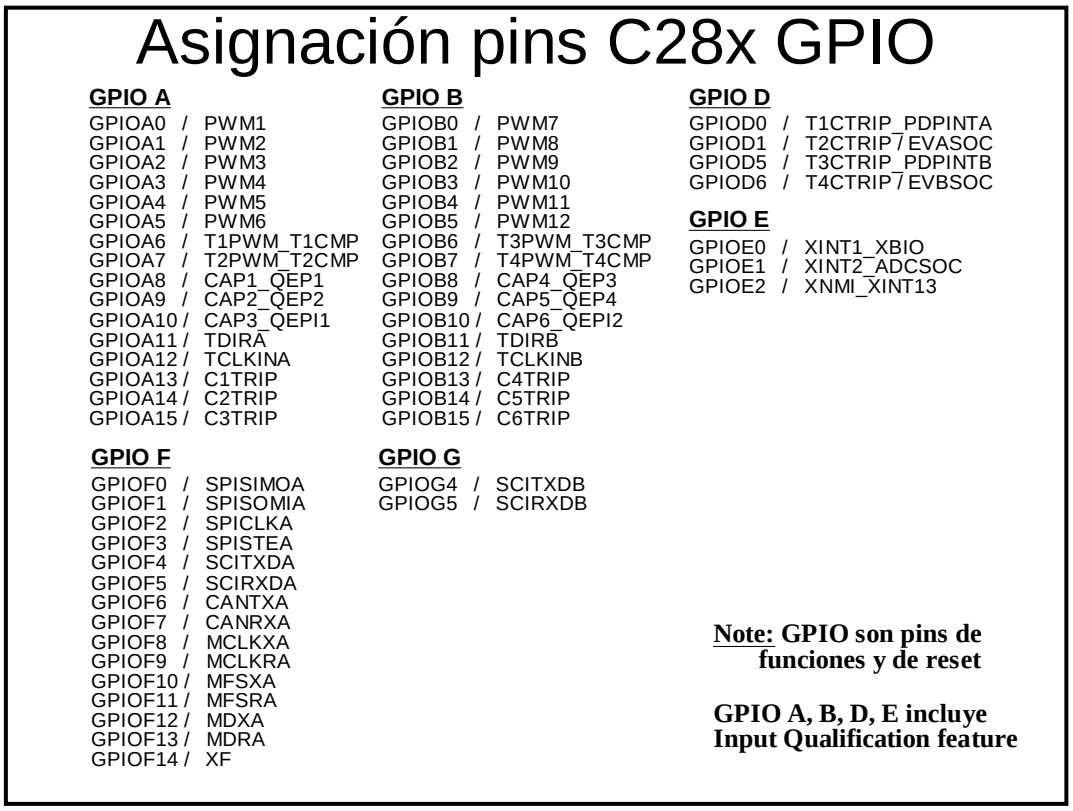


Figura 4.3: Asignación de la función principal o secundaria de los puertos GPIO

El Término “Input Qualification feature” referido a una opción adicional para las señales de entrada digitales a los puertos A, B, D y E. Cuando es utilizada, el pulso de entrada debe ser más largo que el especificado en el número de ciclos para que sea reconocido como una señal de entrada válida.

La siguiente diapositiva explica el procedimiento de inicialización. Los seis puertos GPIO son controlados por nuestro registro multiplex, llamado GPxMUX (donde x puede ser desde A hasta F). Poner una posición de un bit a 0 significa seleccionar su función digital de I/O, poner un bit a 1 significa seleccionar la función especial. (TI llama a esto las “primary” function, función primaria).

Cuando una señal digital I/O es seleccionada, se registra en el grupo GPxDIR definiendo la dirección del I/O (Figura 4.4). Poniendo el bit a 0 se configura la línea como una entrada, sin embargo poniéndola a 1 la configuras como una salida (Figura 4.5). Muchos de los puertos de entrada estén equipados con un "Input Qualification Feature". Con esta opción nosotros podemos definir la duración del tiempo, que es usado para excluir pulsos de una duración mas corta que la del reconocimiento como señales de entrada válidas.

Figura 4.4: Estructura de los registros de los puertos GPIO

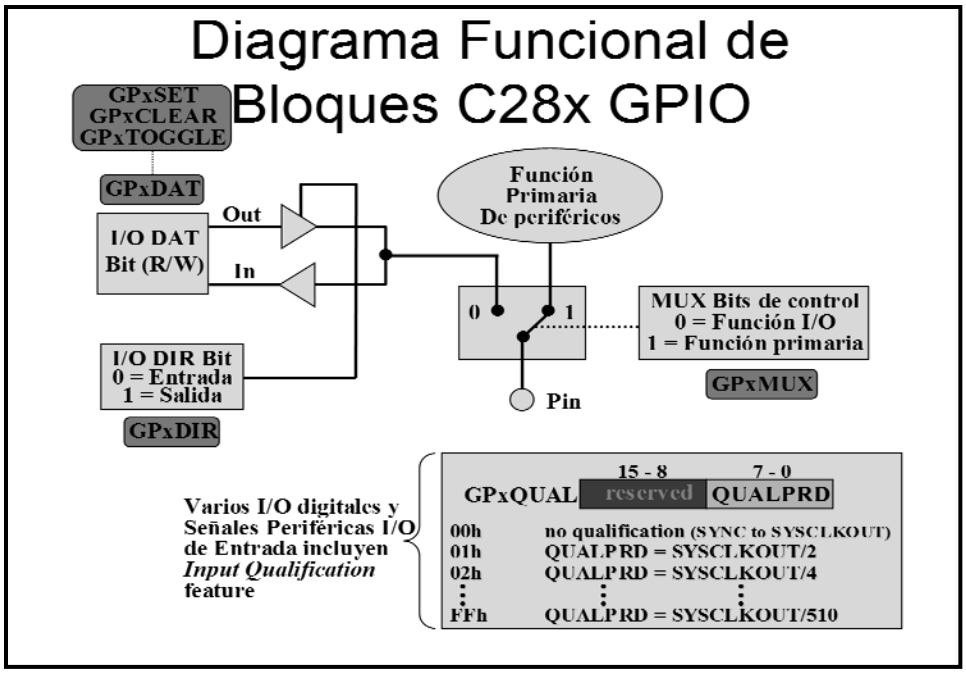


Figura 4.5: Diagrama funcional para la configuración de los GPIO

1.4.5.- Registros Digitales I/O

Las siguientes tablas resumen los comandos de los diferentes puertos de entradas/salidas digitales:

Registros C28x GPIO MUX/DIR

Address	Register	Name
70C0h	GPAMUX	GPIO A Mux Control Register
70C1h	GPADIR	GPIO A Direction Control Register
70C2h	GPAQUAL	GPIO A Input Qualification Control Register
70C4h	GPBMUX	GPIO B Mux Control Register
70C5h	GPBDIR	GPIO B Direction Control Register
70C6h	GPBQUAL	GPIO B Input Qualification Control Register
70CCh	GPDMUX	GPIO D Mux Control Register
70CDh	GPDDIR	GPIO D Direction Control Register
70CEh	GPDQUAL	GPIO D Input Qualification Control Register
70D0h	GPEMUX	GPIO E Mux Control Register
70D1h	GPEDIR	GPIO E Direction Control Register
70D2h	GPEQUAL	GPIO E Input Qualification Control Register
70D4h	GPFMUX	GPIO F Mux Control Register
70D5h	GPFDIR	GPIO F Direction Control Register
70D8h	GPGMUX	GPIO G Mux Control Register
70D9h	GPGDIR	GPIO G Direction Control Register

Tabla 4.6: Registros de los GPIO para los comandos MUX y DIR

C28x GPIO Registros de Datos

Address	Register	Name
70E0h	GPADAT	GPIO A Data Register
70E1h	GPASET	GPIO A Set Register
70E2h	GPACLEAR	GPIO A Clear Register
70E3h	GPATOGGLE	GPIO A Toggle Register
70E4h	GPBDAT	GPIO B Data Register
70E5h	GPBSET	GPIO B Set Register
70E6h	GPBCLEAR	GPIO B Clear Register
70E7h	GPBTOGGLE	GPIO B Toggle Register
70ECh	GPDDAT	GPIO D Data Register
70EDh	GPDSET	GPIO D Set Register
70EEh	GPDCLEAR	GPIO D Clear Register
70EFh	GPDTOGGLE	GPIO D Toggle Register
70F0h	GPEDAT	GPIO E Data Register
70F1h	GPESET	GPIO E Set Register
70F2h	GPECLEAR	GPIO E Clear Register
70F3h	GPETOGGLE	GPIO E Toggle Register
70F4h	GPFDAT	GPIO F Data Register
70F5h	GPFSET	GPIO F Set Register
70F6h	GPFCLEAR	GPIO F Clear Register
70F7h	GPFTOGGLE	GPIO F Toggle Register
70F8h	GPGDAT	GPIO G Data Register
70F9h	GPGSET	GPIO G Set Register
70FAh	GPGCLEAR	GPIO G Clear Register
70FBh	GPGTOGGLE	GPIO G Toggle Register

Tabla 4.7: Registros de los GPIO para los comandos DATA, SET, CLEAR y TOGGLE.

1.4.6.- Modulo de reloj de C28x

Antes de comenzar a describir las entradas/salidas digitales, es necesario configurar el módulo de reloj de C28x (Figura 4.8). Como todos los procesadores modernos, el C28x esta temporizado por un oscilador externo más lento para reducir el ruido electromagnético. Un circuito interno de PLL genera la velocidad interna. El eZdsp funciona con un reloj externo de 30MHz. Para alcanzar la frecuencia interna de 150 MHz hay que utilizar un multiplicador por 10 y el factor se divide por 2. Esto se puede hacer programando el PLL con el registro de control (PLLCR).

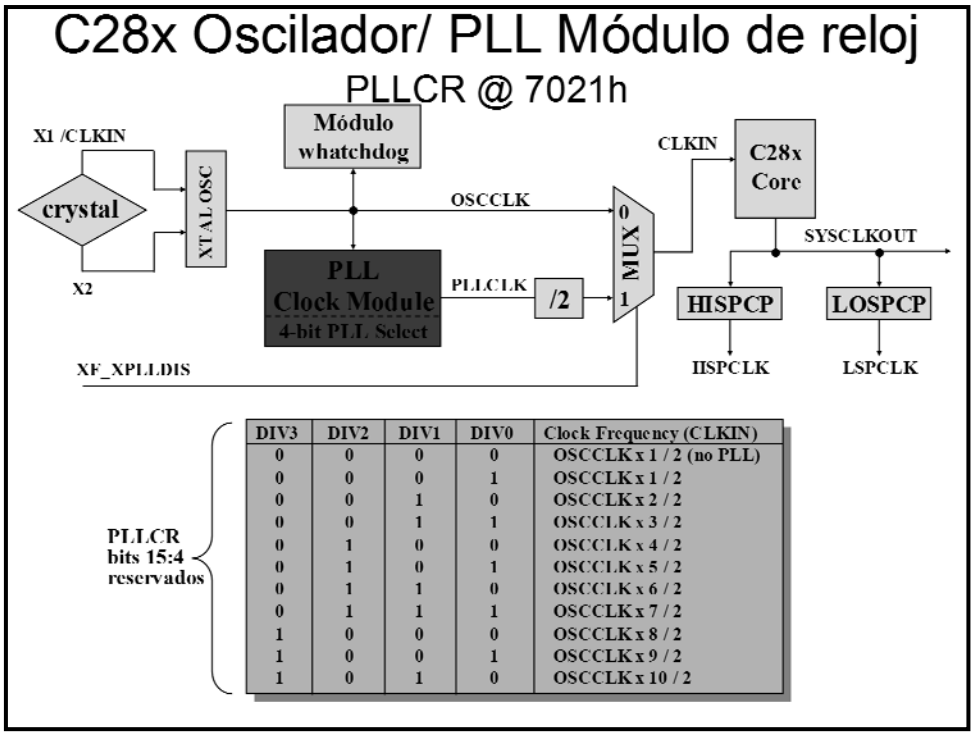


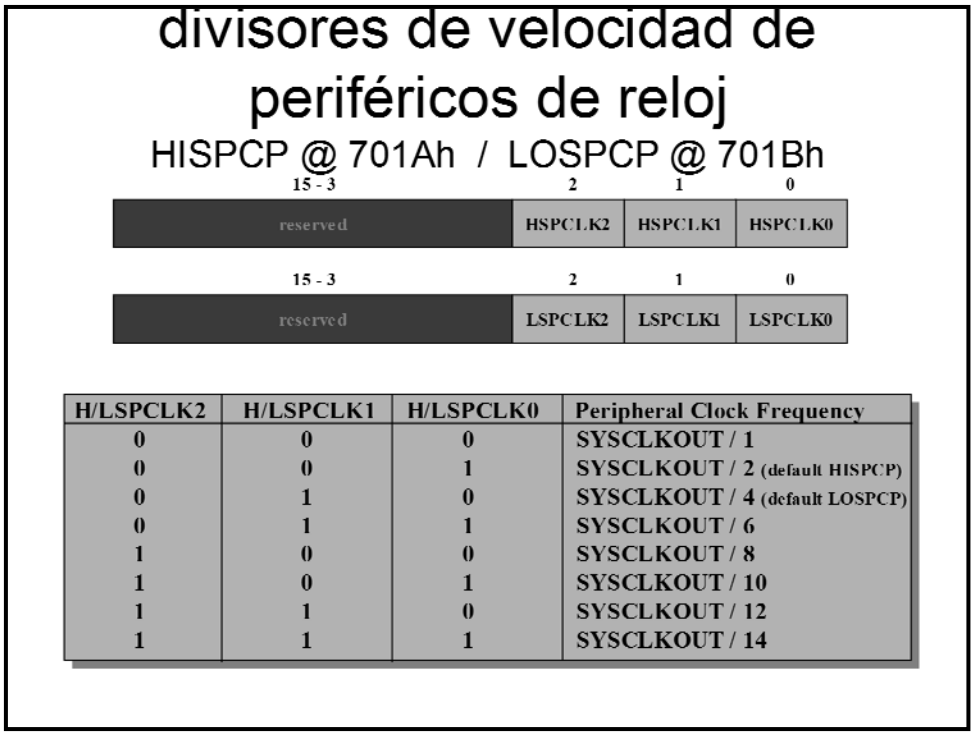
Figura 4.8: Modulo de configuración del reloj.

El Predivisor de alta velocidad del Reloj (HISPCP) y se utiliza el predivisor bajo del reloj de velocidad (LOSPCP) como un divisor adicional de reloj (Figura 4.7). Las salidas de los dos predivisores se utilizan como la fuente del reloj para unidades periféricas. Podemos fijar encima de los dos predivisores individualmente a nuestras necesidades.

Obsérvese que la señal "CLKIN" es de la misma frecuencia que la señal de salida de base "SYSCLKOUT", se utiliza que para el interfaz externo de la memoria y para registrar la unidad CAN.

También, que la unidad del perro guardián es registrada directamente por el oscilador externo.

Finalmente, que la frecuencia máxima para el oscilador externo es 35MHz.



H/LSPCLK2	H/LSPCLK1	H/LSPCLK0	Peripheral Clock Frequency
0	0	0	SYSCLKOUT / 1
0	0	1	SYSCLKOUT / 2 (default HISPCP)
0	1	0	SYSCLKOUT / 4 (default LOSPCP)
0	1	1	SYSCLKOUT / 6
1	0	0	SYSCLKOUT / 8
1	0	1	SYSCLKOUT / 10
1	1	0	SYSCLKOUT / 12
1	1	1	SYSCLKOUT / 14

Figura 4.9: Divisores de la frecuencia del reloj.

Para utilizar una unidad periférica, podemos permitir su distribución de reloj por los campos individuales del bit del registro PCLKCR (Figura 4.10). La entrada/salida digital no hace que un reloj permita esta característica.

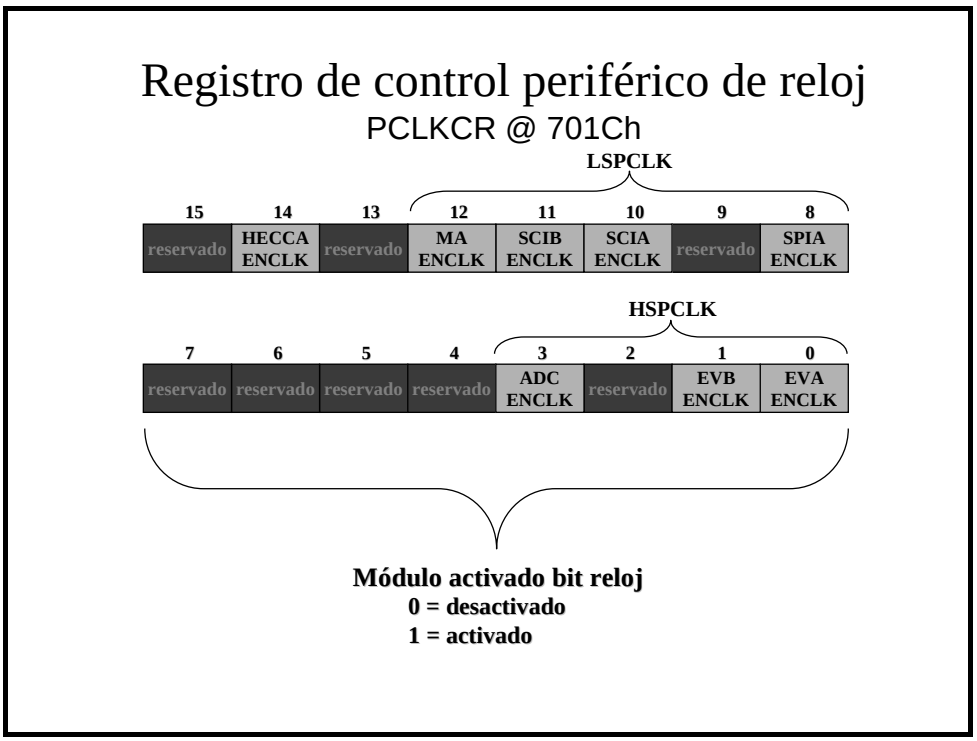


Figura 4.10: Control periférico del reloj.

1.4.7.- Contador Perro Guardián

El "contador de tiempo perro guardián" es una unidad libre de contador que resetea el programa si no se borra periódicamente mediante una instrucción especifica. Esto es usado para reconocer eventos donde el programa permite que sea designada una secuencia de ejecución, por ejemplo, cuando el programa se estropea.

Las características de este contador son las que aparecen en la figura 4.11 y el modulo de configuración es el recogido en la figura 4.12.

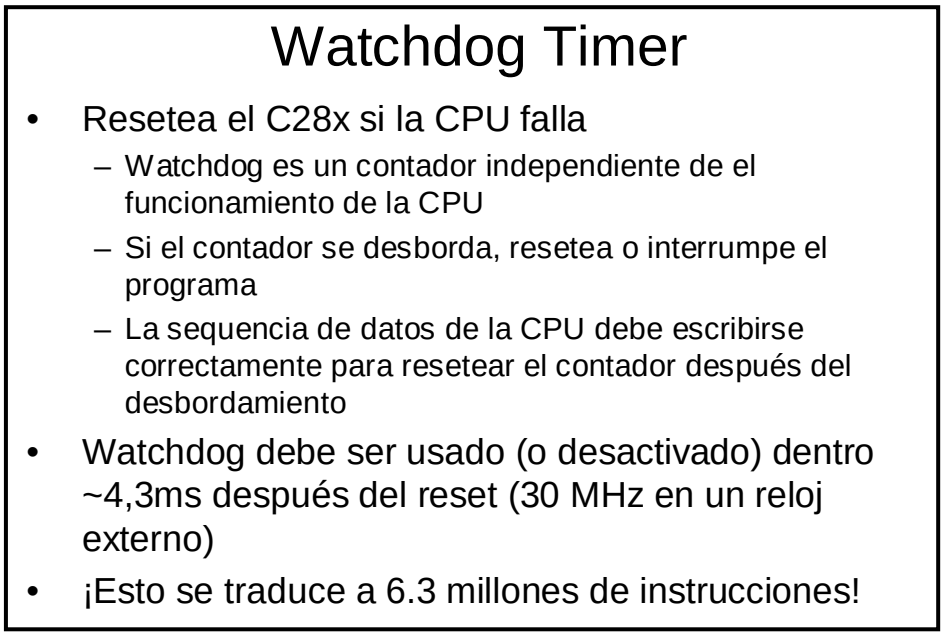


Figura 4.11: Características del perro guardián.

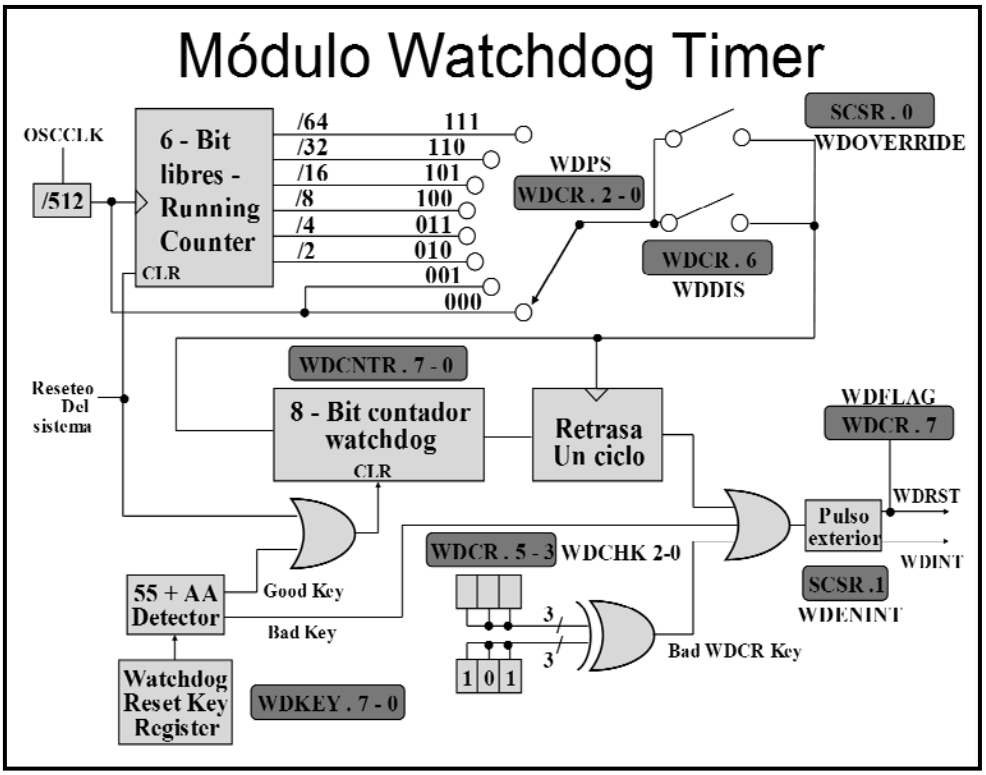


Figura 4.12: Modulo de configuración del perro guardián.

El perro guardián está siempre activo mientras el DSP está conectado. Cuando no lo reiniciamos periódicamente, el perro guardián resetea el DSP. Uno de los métodos más simples para apagarlo es mediante el bit 6 (WDFLAG), poniéndolo a 1. Por supuesto esto no es una buena decisión, ya que es un sistema de protección del DSP.

El divisor de frecuencias puede ser usado para el perro guardián para incrementar los periodos a los que se desborda. El Logic Check Bits (WDCHK) es otro bit de seguridad. Todos los accesos para escribir al registro WDCR (Figura 4.13) deben incluir la combinación de bits “101”, si no es así se niega el acceso y se resetea inmediatamente.

El bit de flag del perro guardián (WDFLAG) puede ser usado para distinguir entre un power normal en reset (WDFLAG = 0) y un reset por el perro guardián (WDFLAG = 1). Nota: Para borrar este flag mediante el software debemos escribir un 1 dentro de este bit.

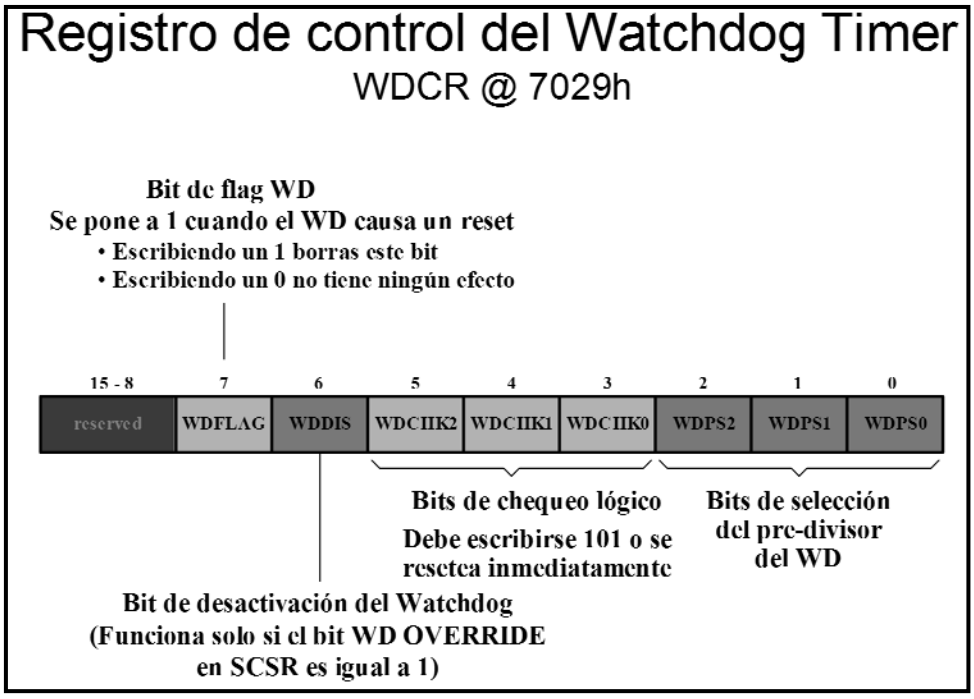


Figura 4.13: Registros de control del perro guardian.

Nota: Si, por alguna razón, el oscilador externo del reloj fallara, el perro guardián pararía el incremento. En este caso podemos tratar esta situación leyendo el registro del perro guardián periódicamente. En caso de que el reloj externo este roto este registro no se incrementará más. El C28x todavía se ejecutaría en modo PLL, puesto que el PLL hace salir un reloj entre 1 y 4 megaciclos en un llamado "limp"-modo.

¿Cómo podemos borrar el perro guardián? Escribiendo una secuencia valida dentro del registro WDKEY como se puede ver en las figuras 4.14 y 4.15:

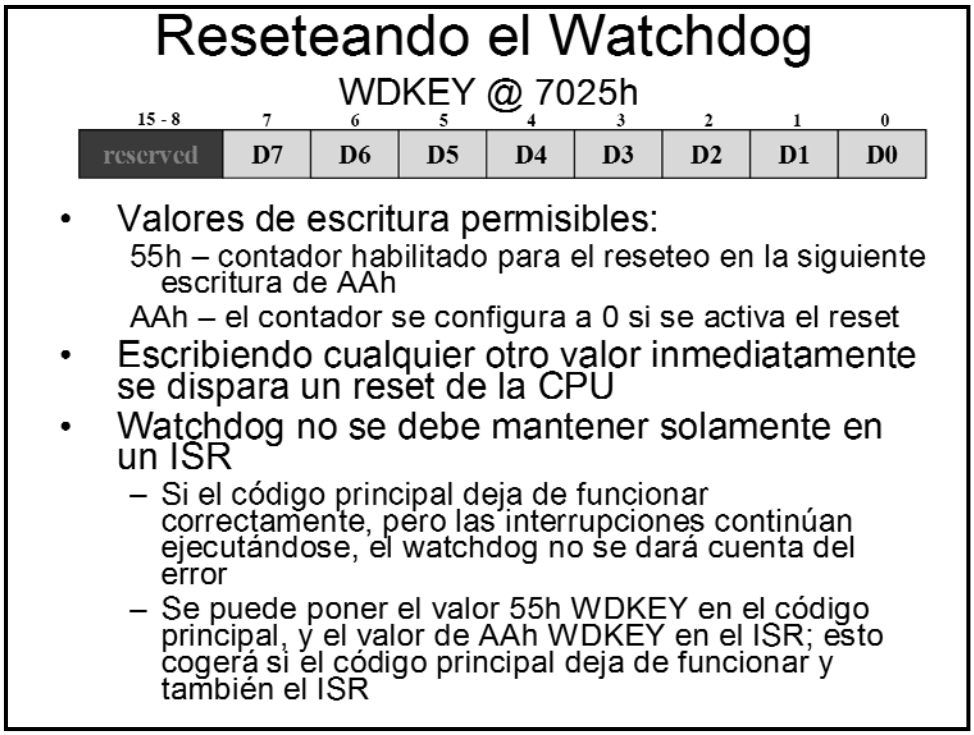


Figura 4.14: Reseteo del perro guardián.

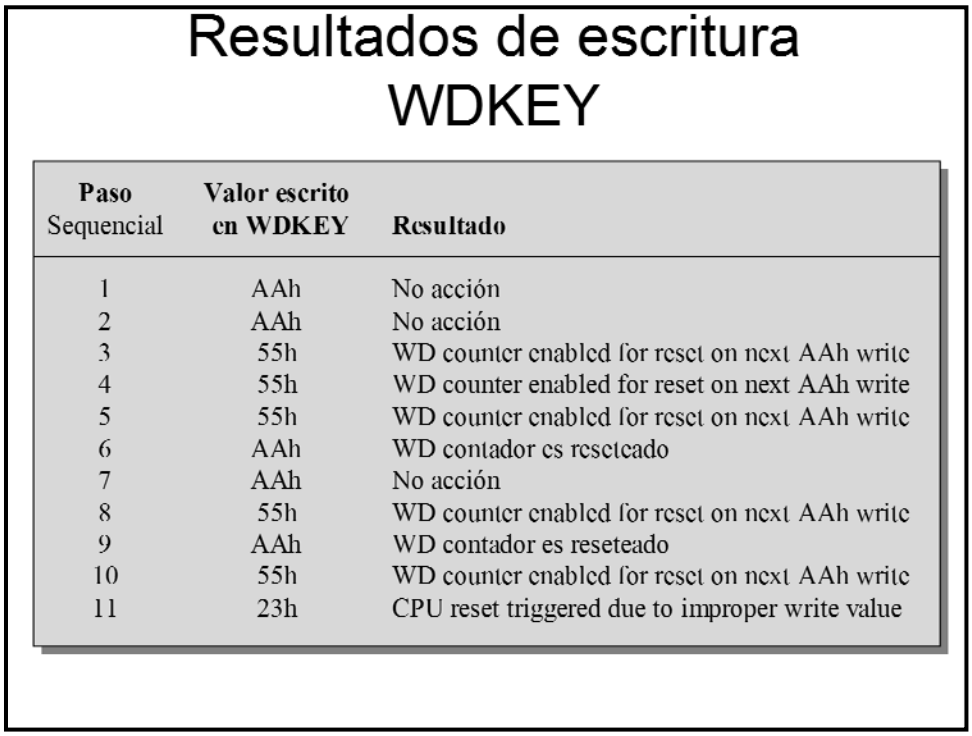


Figura 4.15: Escritura en el registro WDKEY.

1.4.8.- Control de sistema y registro de estado

El registro SCSR controla cuando el perro guardián causa un RESET (WDENINT = 0) o un servicio de interrupción (WDENINT = 1).

El bit WDOVERRIDE es un "clear only" bit, que significa que una vez tengamos cerrado este interruptor escribiendo un 1 en el bit, no podremos volver a abrir este interruptor otra vez (observe el diagrama de bloques del perro guardián). En este punto el bit WD-disable es inefectivo, no hay vía para inhabilitar el perro guardián.

El bit 2 (WDINTS) es un bit de sólo lectura, es el estado de la interrupción del perro guardián.

Todos estos registros y bits se reflejan a continuación:

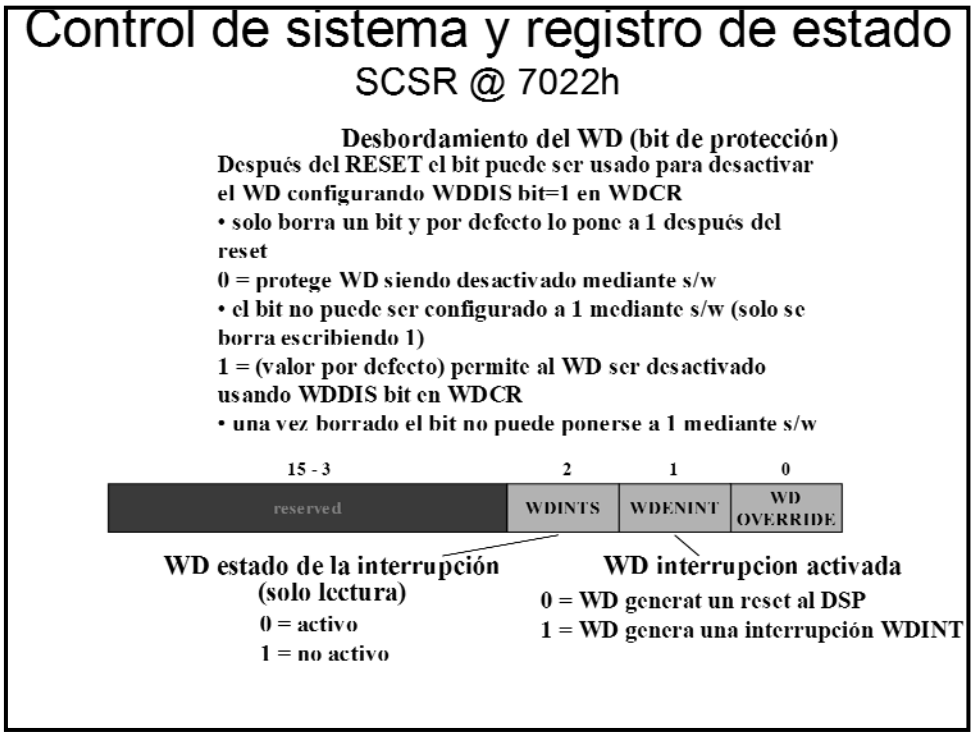


Figura 4.16: Registro de estado del perro guardián.

1.4.9.- Modo en baja potencia

Para reducir el consumo del C28x puede cambiar entre 3 diferentes modos de operación de bajo consumo. No. El modo de bajo consumo está completamente dedicado en la instrucción en ensamblador "IDLE". Mientras no se ejecute esta instrucción en la inicialización del registro LPMCR0 esta no tendrá efecto.

Las siguientes tablas y figuras explican los modos de bajo consumo en detalle:

Modo de baja potencia

Modo de baja potencia	Reloj lógico De la CPU	Reloj logico preifericos	Reloj watchdog	PLL / OSC
Normal Run	on	on	on	on
IDLE	off	on	on	on
STANDBY	off	off	on	on
HALT	off	off	off	off

Tabla 4.17: Estado de los relojes dependiendo del modo de funcionamiento.

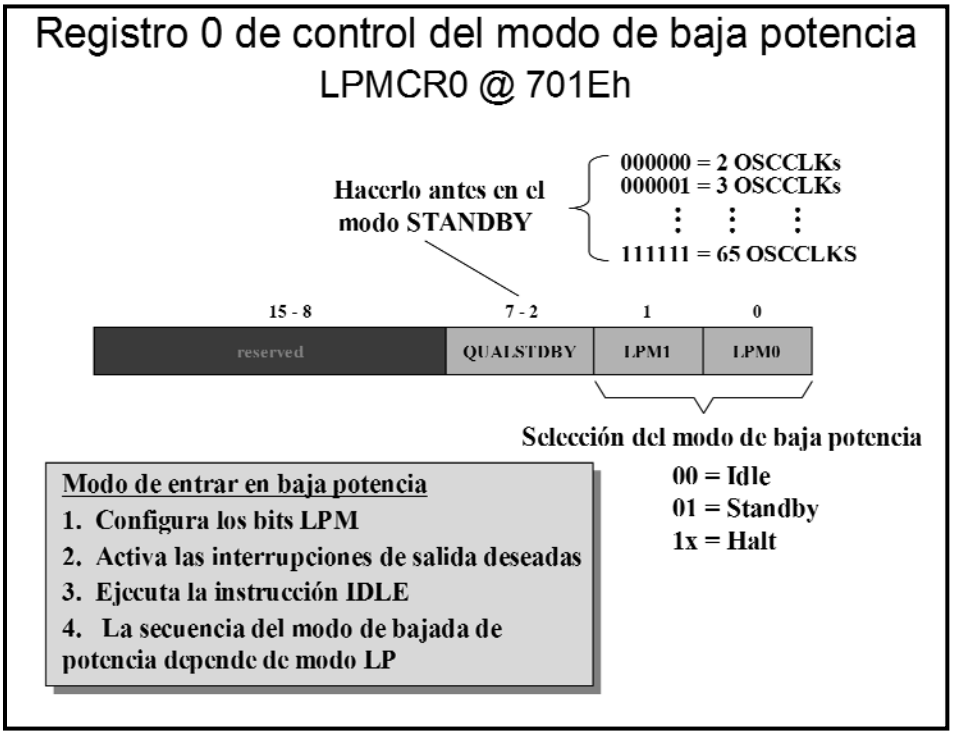


Figura 4.18: Registro 0 del control del modo de baja potencia.

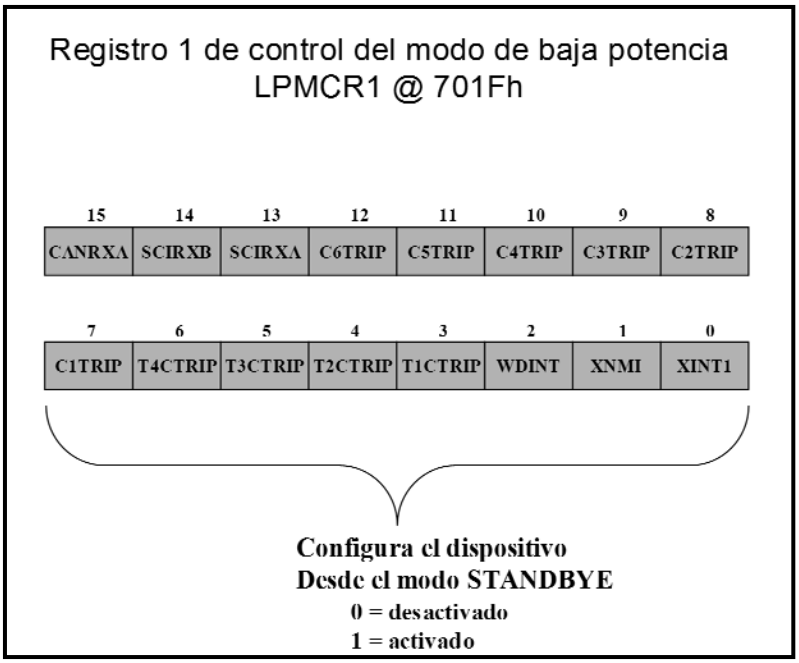


Figura 4.19: Registro 1 del control del modo de baja potencia.

Modo de salida de la baja potencia

Salida de la interrupcion / Modo de Baja potencia	RESET	Encendido externo o interrupciones	Interrupciones Periféricas acitvadas
IDLE	yes	yes	yes
STANDBY	yes	yes	no
HALT	yes	no	no

Nota: Encendido externo o interrupciones incluye XINT1, PDPINT, TxCTRIP, CxTRIP NMI, CAN, SPI, SCI, WD

Tabla 4.20: Salida de las interrupciones dependiendo del modo de funcionamiento.

1.5.- SISTEMA DE INTERRUPCIONES Y TEMPORIZADOR

1.5.1.- Introducción.

En este tema se introduce el concepto de interrupción y se profundiza en las interrupciones del F2812.

Pero, ¿Qué es una interrupción?

Antes de que nos metamos en términos técnicos intentemos comprender el significado de una interrupción por medio de una comparación. Piense que usted está trabajando una bonita mañana en su despacho preparando los experimentos de laboratorio del siguiente día. De repente el teléfono suena, contestas y luego vuelves a tu trabajo (después de tu interrupción). La corta llamada telefónica es más importante. Por supuesto si te llama tu pareja debes pensar bien cual es la prioridad de la interrupción….Sea como sea, mas tarde o mas temprano volverás a tu trabajo.

Esta analogía tiene las definiciones más básicas de las interrupciones:

- Las interrupciones aparecen “de repente”: esto en lenguaje técnico es llamado asincrónicamente.
- Las interrupciones pueden ser más o menos importantes, esto es lo que llamamos “prioridades”.
- Las interrupciones deben ser tratadas antes de que el teléfono pare de sonar, es decir “inmediatamente”.
- La preparación de tus prácticas de laboratorio pueden ser continuadas después de la llamada.
- El tiempo que empleas mientras el teléfono está sonando debe ser lo mas corto posible.
- Después de la llamada se deberá continuar con el trabajo en la misma posición donde lo habías dejado.

Para resumir los términos técnicos:

Las interrupciones pueden ser llamadas eventos asíncronos, generados por una unidad externa o interna. Estas interrupciones hacen que el DSP se vaya a la subrutina de interrupciones que esta dedicada a ese evento. Después de ejecutar esa subrutina se volverá a su programa principal, justo donde se produjo la interrupción.

La rapidez con la que la CPU ejecuta estas interrupciones es lo más importante para el control en tiempo real. Después de este capítulo usted será capaz de entender el sistema de interrupciones del C28x.

1.5.2.- Líneas de interrupción del núcleo de C28x

Estas consisten en 16 líneas de interrupción (Figura 5.1), dos de ellas son “no enmascarables” (RESET, NMI). Las otras 14 líneas serán “enmascarables”, esto significará que el programador podrá trabajar con 14 líneas de interrupción.

Pero, ¿Qué significa “enmascarable” o “no enmascarable”?

Un “enmascaramiento” es una combinación binaria de uno y cero. El uno significará que la línea de interrupción esta activa, usaremos el 0 para desactivarla. Mediante la carga del enmascaramiento en el registro “IER” podremos seleccionar que líneas de interrupción queremos que estén activas y cuales no.

Para una “no enmascarable” podremos denegar la petición de una interrupción. Cuando la línea de señal esté activa, el funcionamiento del programa podrá ser parado y dedicarle una subrutina de tratamiento de las interrupciones cuando empiece.

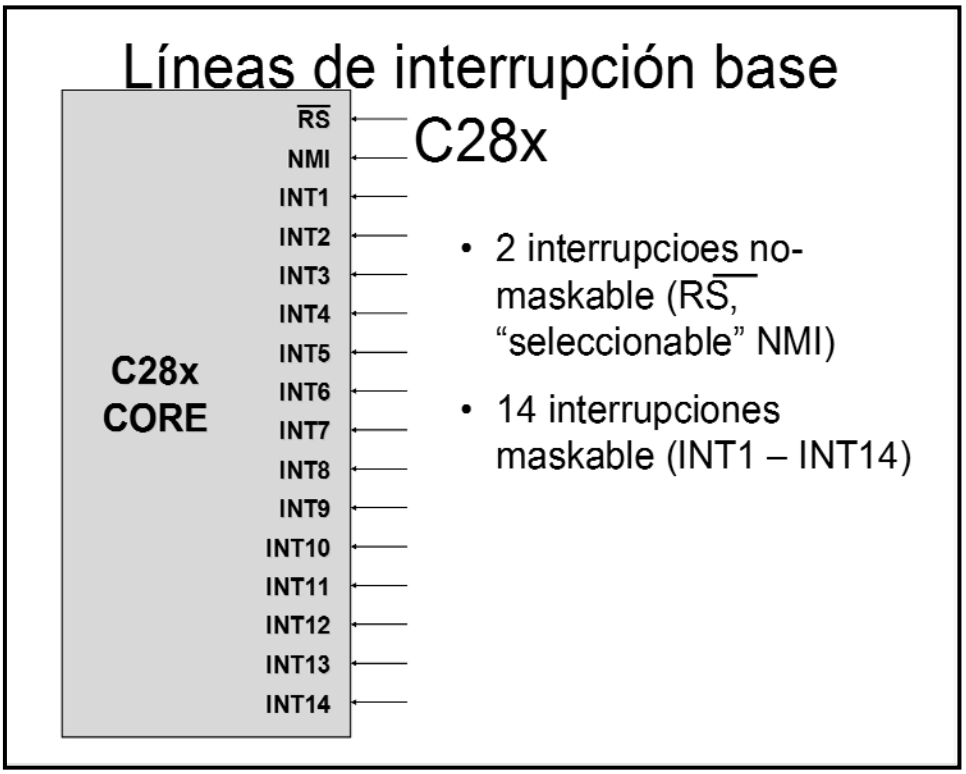


Figura 5.1: Resumen de las líneas de interrupción del C28XX

Todas las 16 líneas están conectadas a una tabla de vectores de interrupción, que consisten en 32 bits de memoria de localización para las interrupciones. Es responsabilidad del programador rellenar esta tabla cuando quiera usar la rutina de servicio de interrupciones.

1.5.3.- Reset del C28x

Un cambio de 1 a 0 en el pin "/RS" causa el reset del DSP. Este evento fuerza al DSP a empezar desde esta dirección de reset (Código de memoria 0x3F FFCO). Este evento no es una interrupción propiamente dicha, se genera un reset al arrancar el DSP.

Otra manera de resetear el DSP es mediante el desbordamiento del perro guardián. Para informar al DSP de que se ha desbordado, este produce un reset, el DSP por si solo controla este pin. Esto significa que este pin puede ser bidireccional (Figura 5.2).

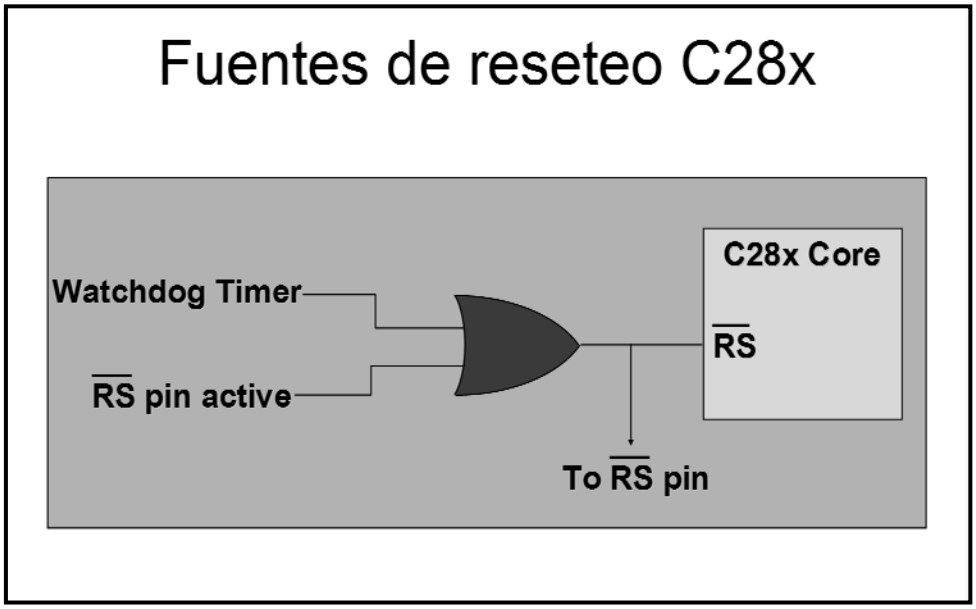


Figura 5.2: Causas que producen un reset

El reset fuerza al DSP a comenzar de nuevo desde la dirección 0x3F FFC0, esto inicializa todas las operaciones de registros internos, resetea los registros de Flags a sus estados iníciales y desactiva las 16 líneas de interrupción.

Como se comentó en el capítulo 5 de este proyecto, el mapa de memoria del C28x permite tener dos memorias físicas con las direcciones 0x3F FFC0 que son la memoria interna y la memoria externa. Con el pin XMPMC se decide el modo de funcionamiento. Configurando este pin a 1 se selecciona la memoria externa y se desactiva la dirección interna. Conectando este pin a 0 sucede al contrario, es decir, se selecciona la memoria interna ROM para ser usada como una dirección de memoria. El estado del bit XMP/MC será copiado en el pin 'XMP/MC' que podrá ser usado por el software más tarde.

1.5.4.- Reset Bootloader

Cuando se selecciona la memoria interna, el software precargado, denominado BootLoader arranca. En función del estado de cuatro pines del chip (Tabla 5.1) el proceso de arranque puede producirse de diferentes modos (Figura 5.3). Estos pines son configurados en la placa eZdsp con los puertos JP1, JP7, JP8, JP11, y JP12.

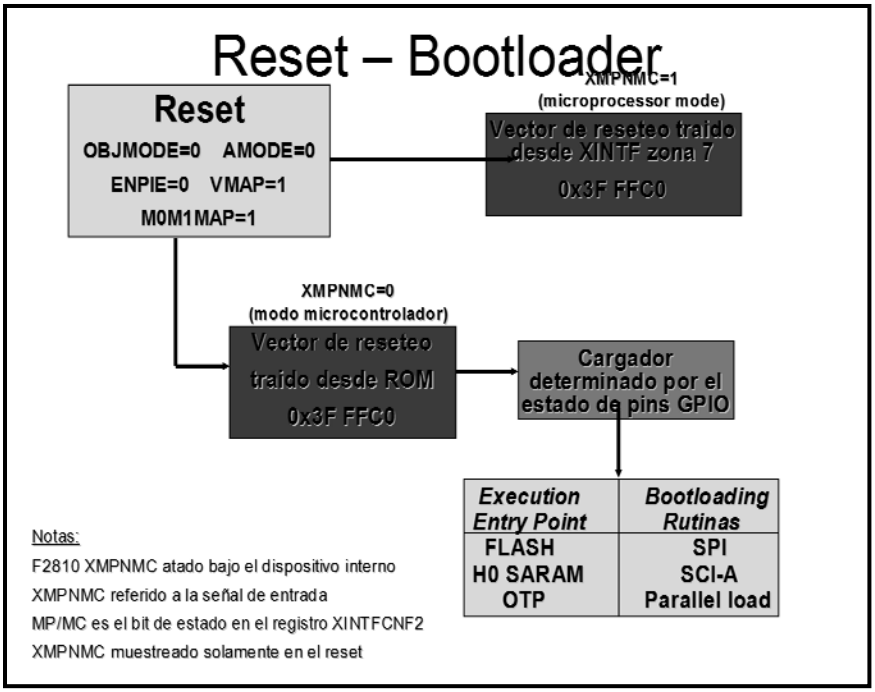


Figura 5.3: Modos de arranque cuando se usa memoria interna.

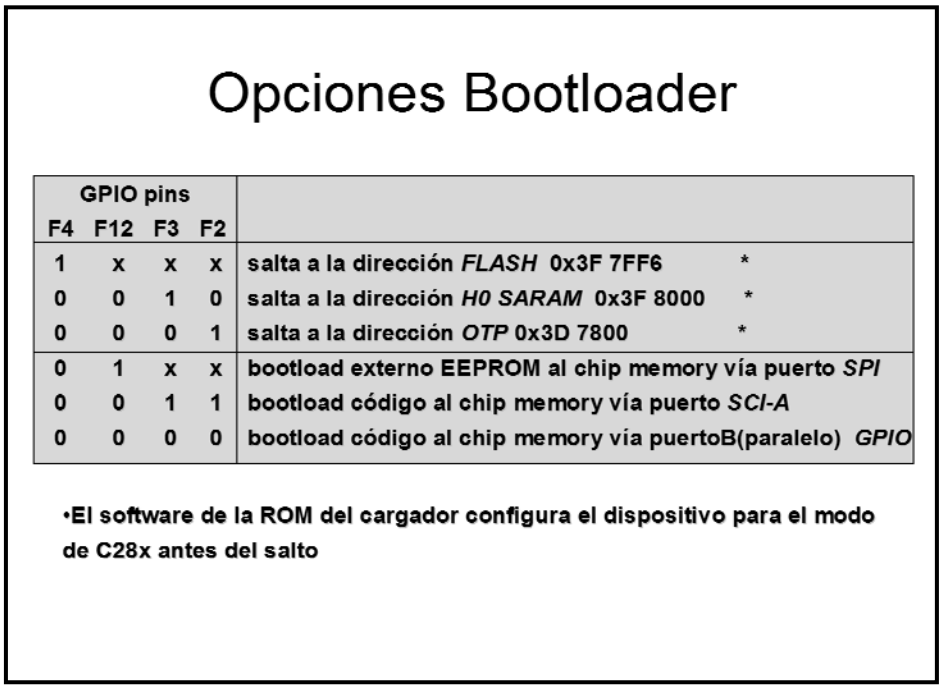


Tabla 5.4: Salidas producidas en función de los pines del Bootloader.

El modo mas normal de arranque para emplear las placas en el laboratorio es el que salta a la dirección del bloque 0x3F8000 en el bloque H0 de la SDRAM. A continuación se detalla de forma gráfica el salto que hace el DSC cuando arranca y lo que significa cada una de las opciones de arranque.

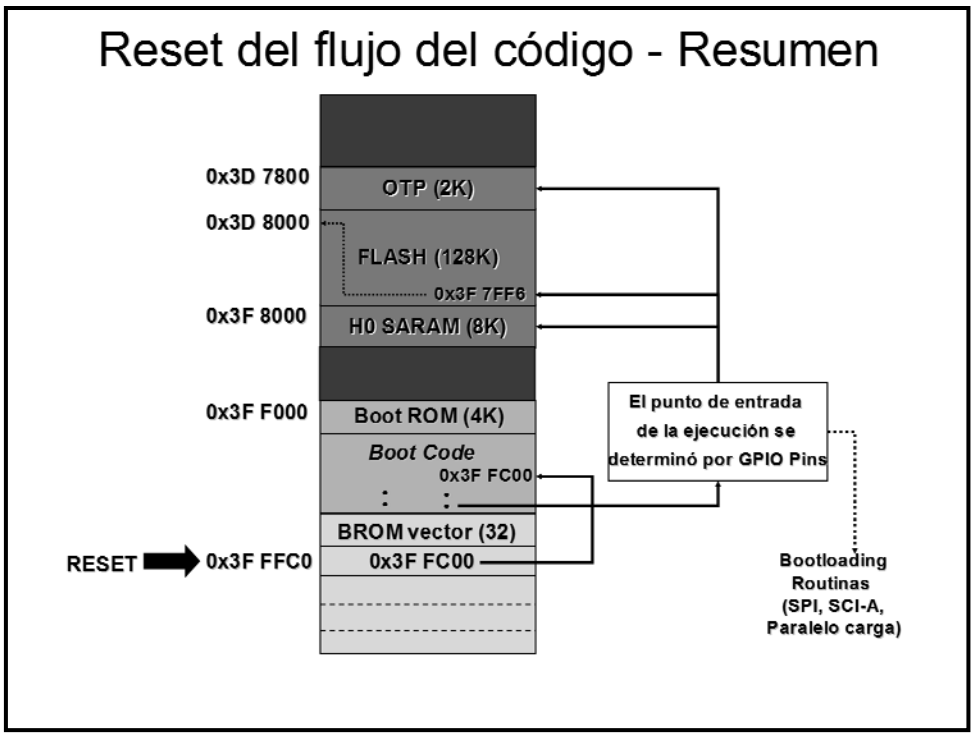


Figura 5.5: Salto a la direccion 0x3F8000 que se produce al arrancar el Dsp

La opción 'Flash Entry' normalmente se usa al final de cada fase de desarrollo del proyecto cuando el sofware está libre de errores. Para cargar el programa en la flash se necesita un programa específico, accesible como cualquier plugin de Code Composer Studio o por separado.

Las opciones de los puertos de interfaz serie (SPI / SCI) o paralelo de BootLoader son usadas para descargar el código ejecutable desde un ordenador externo o para mejorar los contenidos de la memoria flash. Nosotros no usaremos estas opciones en este tutorial.

La memoria OTP es programable solo una vez en el tiempo, es decir no hay una segunda oportunidad de rellenar el código en esta memoria no volátil. Esta opción es usada normalmente por las compañías que producen código y así proteger la propiedad intelectual.

1.5.5.- Fuentes de Interrupción

Las fuentes de interrupción que tiene este integrado son numerosas. (96 de momento) pero la CPU solo tiene 14 líneas de interrupción de las cuales dos son interrupciones de entrada 'enmascarables'. La cuestión es: ¿Cómo se produce esta reducción de 96 a 14?. Se explicará mas adelante aunque es obvio que por cada línea de interrupción circularán algunas de las 96 posibles fuentes (Figura 5.6).

Cada una de las líneas de interrupción está asociada a un vector de interrupción, que es una posición de memoria de de 32 bits dentro de una zona denominada tabla de vectores de interrupción. Este espacio de memoria almacena la dirección de la rutina de servicio de esa interrupción, denominada ISR (Interrupt Service Rutine). En el caso que varias fuentes de interrupción activen la misma rutina ISR, el programador deberá discriminar de alguna forma que fuente ha generado la interrupción. Este método consume ciclos de reloj y puede no ser adecuado para las aplicaciones en tiempo real.

Se puede mejorar la rapidez del sevicio incluyendo un gestor de interrupciones por hardware. Es el Peripheral Interrupt Expansion (PIE).

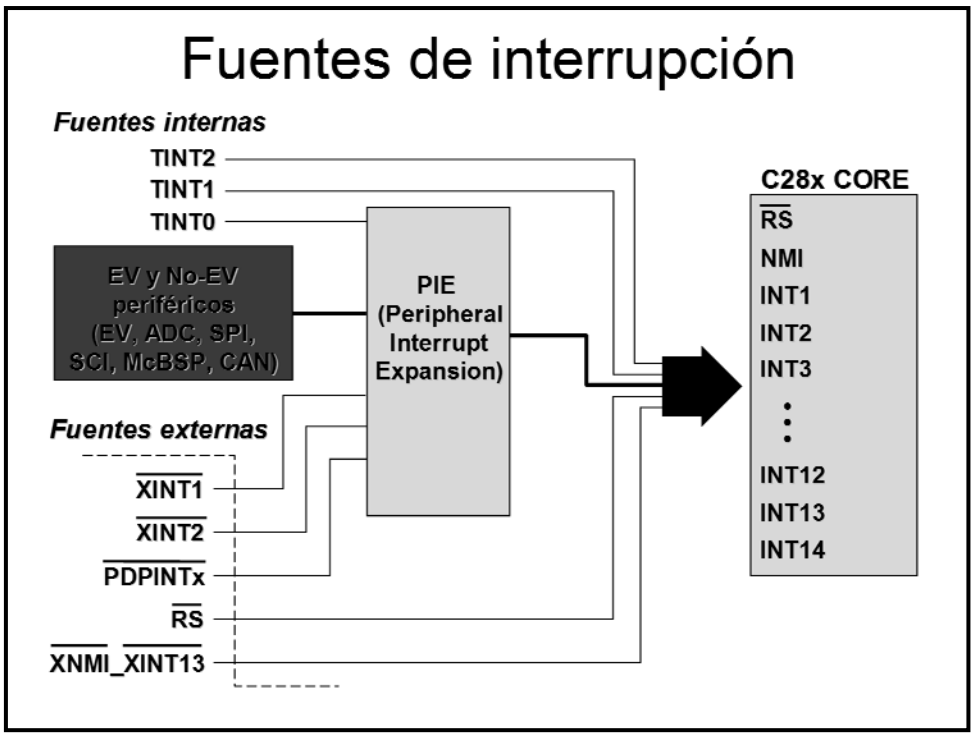


Figura 5.6: Posibles fuentes de interrupción.

Esta unidad aumenta el número de vectores de interrupción, reservando 32 bits individuales para cada una de las 96 interrupciones posibles. De esta forma se consigue una respuesta a la interrupción mucho más rápida. Para usar el PIE hay que remapear la localización de la tabla de vectores de interrupción a la dirección 0x00 0D00, esto es, en la memoria volátil. Previamente hay que reservar esta memoria para inicializar el PIE.

1.5.6.- Proceso de Interrupción Enmascarable

Antes de pasar al ver como funciona el PIE es necesario indicar que camino o trayectoria sigue una señal de interrupción que llega a una línea de interrupción antes de ser reconocida por el núcleo del DSC y acceder a su vector correspondiente. En las siguientes figuras se indica el camino a seguir para poder atender correctamente a una interrupción.

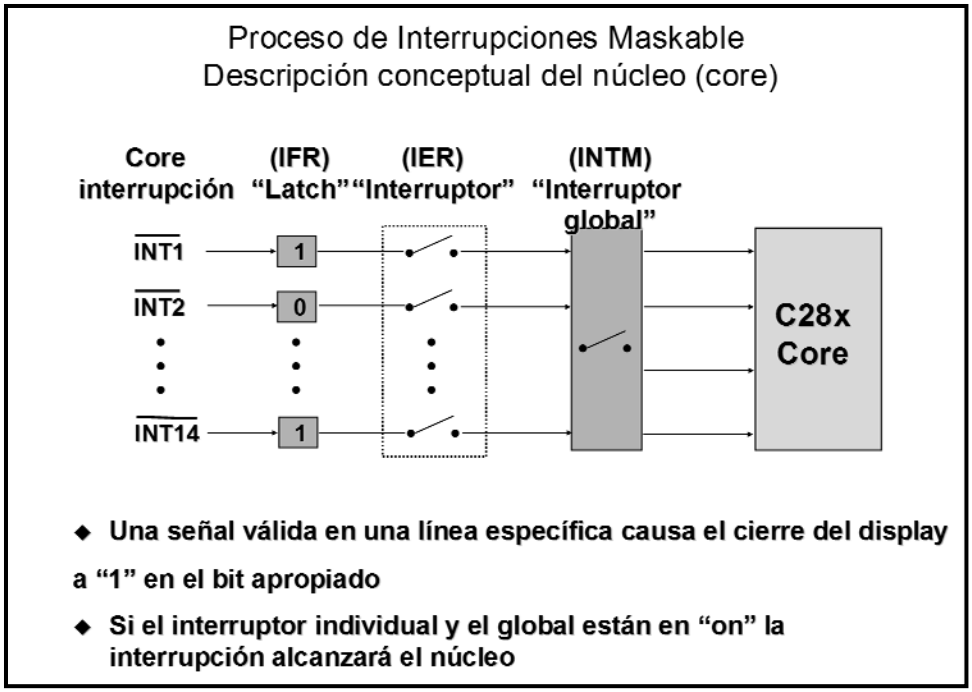


Figura 5.7: Interruptores para poder activar una interrupción enmascarable.

El registro IFR cuya estructura se detalla en la Figura 5.8 es el que contiene los flags de cada interrupción, estos flags se quedan activados cuando se produce una interrupción y deben ser borrados para que puedan haber nuevas interrupciones, normalmente su borrado se lleva a cabo dentro de la rutina de interrupción.

El registro IER cuya estructura se detalla en la Figura 5.9, es el que permite que una interrupción sea atendida, si este registro no tiene activo el bit de la interrupción que queremos atender, esta no producirá nada. Por lo tanto para poder trabajar con una interrupción, debemos activar su bit correspondiente en el registro IER

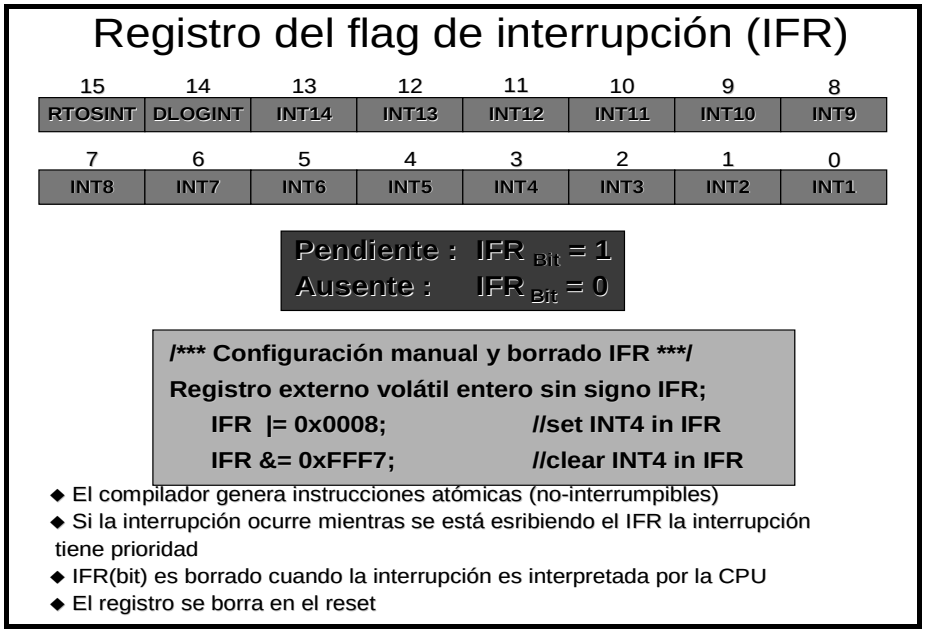


Figura 5.8: Registro de los flags de las interrupciones.

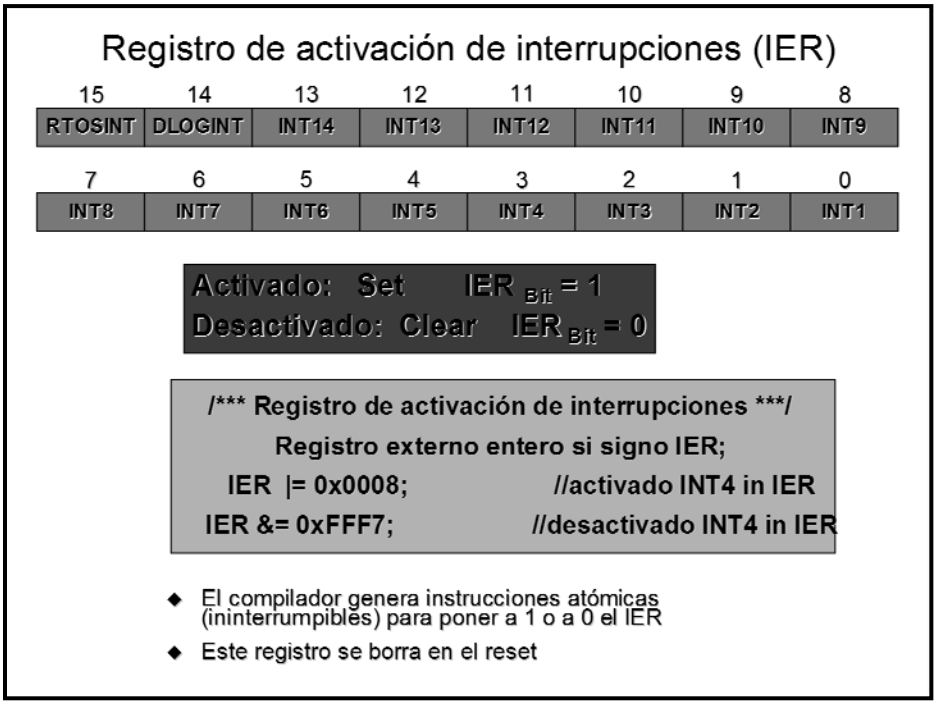


Figura 5.9: Registro de activación de las interrupciones.

El registro INTM, cuya estructura se detalla en la figura 5.10, es el que habilita todas las interrupciones, si este bit no se encuentra activo no se producirá ninguna interrupción.

En resumen: si se va a activar una interrupción, primero se debe activar el bit global de interrupciones (INTM), luego el bit del IER correspondiente a la interrupción que vamos a utilizar y por último se debe borra el flag del IFR para que puedan haber nuevas interrupciones.

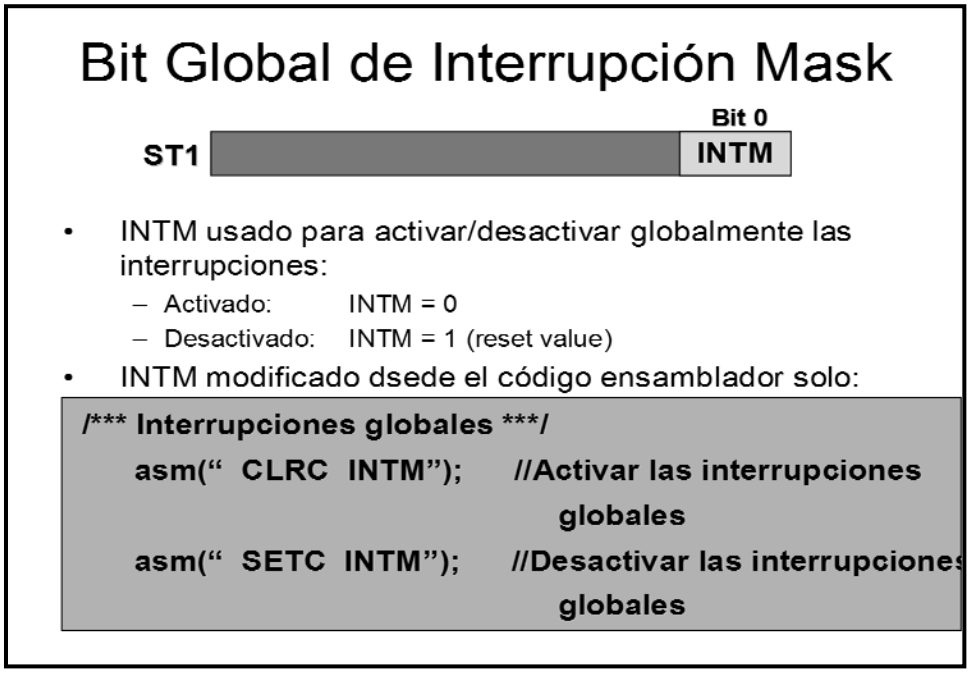


Figura 5.10: Bit global de interrupciones.

1.5.7.- Unidad Periférica de Expansión (PIE)

El PIE gestiona las 96 posibles fuentes de interrupción para que sean dirigidas por las 12 líneas de interrupción no enmascarables existentes (Figura 5.11).

Las 96 posibles fuentes de interrupción están agrupadas en 12 grupos de interrupciones. Cada grupo contiene 8 fuentes, lo que hace un total de 96. Para habilitar o deshabilitar cada código individual tendremos otro grupo de registros: 'PIEIFRx' y 'PIEIERx' (Figura 5.12) que funcionan de forma similar a los IER e IFR vistos anteriormente.

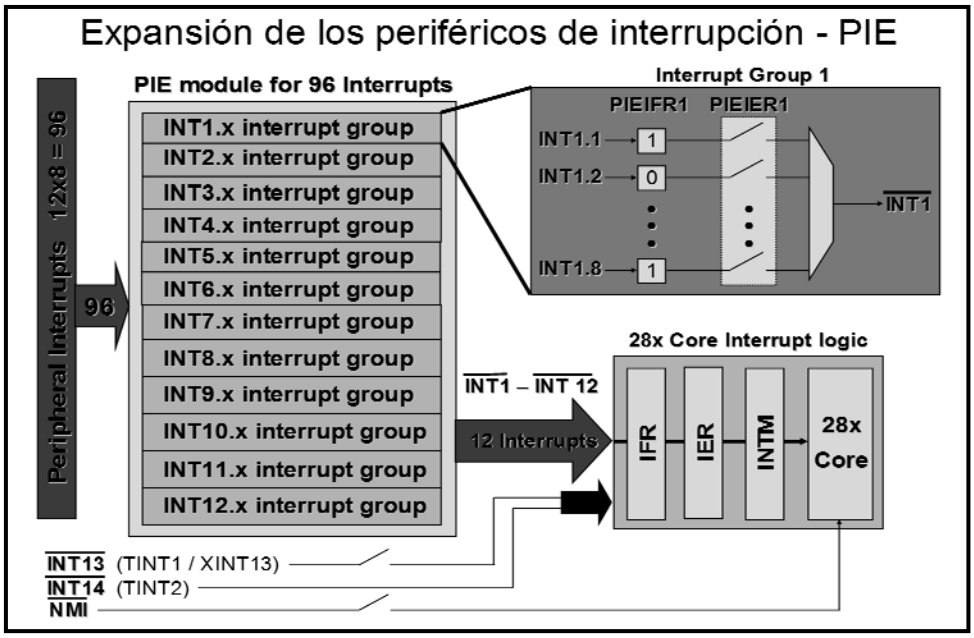


Figura 5.11: Esquema del PIE

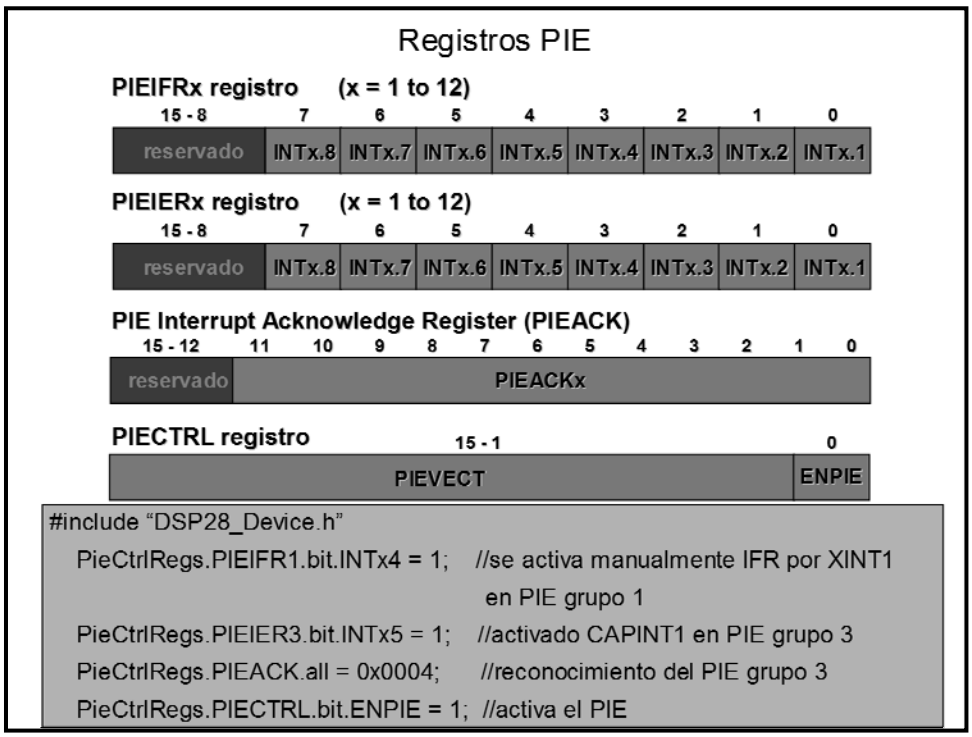


Figura 5.12: Registros del PIE

Todas las fuentes de interrupción están conectadas a las 12 líneas de interrupción de acuerdo con la tabla de asignación siguiente:

F2812/10 PIE Tabla de asignación de interrupciones

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKEINT	TINT0	ADCINT	XINT2	XINT1		PDPINTB	PDPINTA
INT2		T1OFINT	T1UFINT	T1CINT	T1PINT	CMP3INT	CMP2INT	CMP1INT
INT3		CAPINT3	CAPINT2	CAPINT1	T2OFINT	T2UFINT	T2CINT	T2PINT
INT4		T3OFINT	T3UFINT	T3CINT	T3PINT	CMP6INT	CMP5INT	CMP4INT
INT5		CAPINT6	CAPINT5	CAPINT4	T4OFINT	T4UFINT	T4CINT	T4PINT
INT6			MXINT	MRINT			SPITXINTA	SPIRXINTA
INT7								
INT8								
INT9			ECAN1INT	ECAN0INT	SCITXINTB	SCIRXINTB	SCITXINTA	SCIRXINTA
INT10								
INT11								
INT12								

Tabla 5.13: Tabla de asignación de interrupciones.

Para activar una interrupción, por ejemplo la ADCINT, tendremos que poner a 1el bit 6 del grupo 1 : INT1.6.

La tabla de localización de vectores tras el reset será:

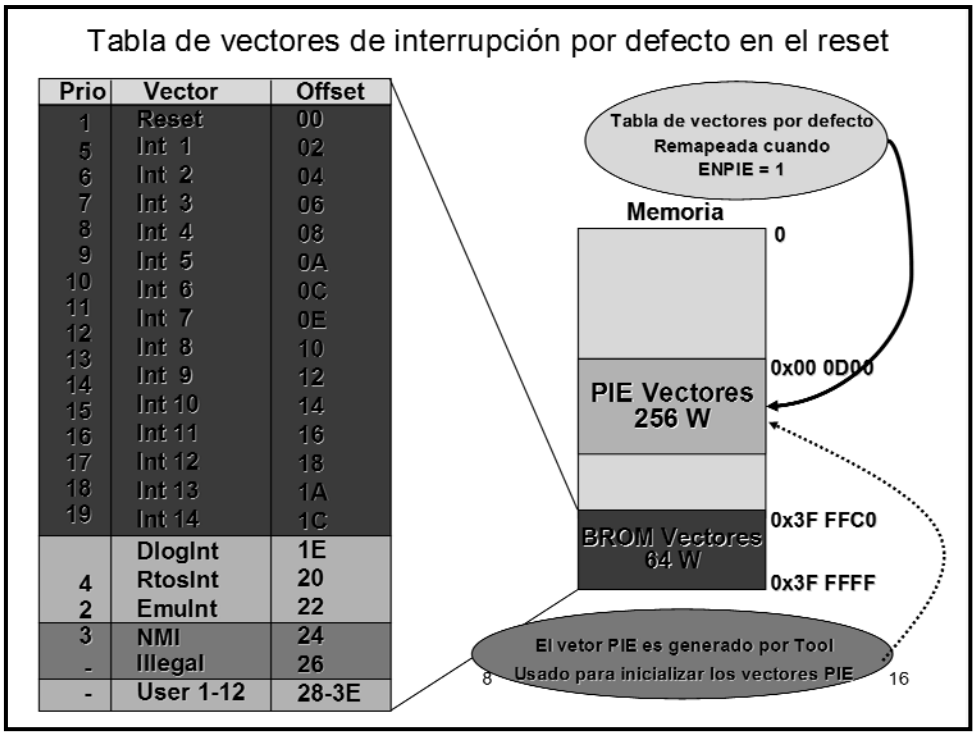


Figura 5.14: Vectores de las interrupciones tras el reset.

El remapeado de los vectores de interrupción cuando se emplea el PIE se muestra ahora:

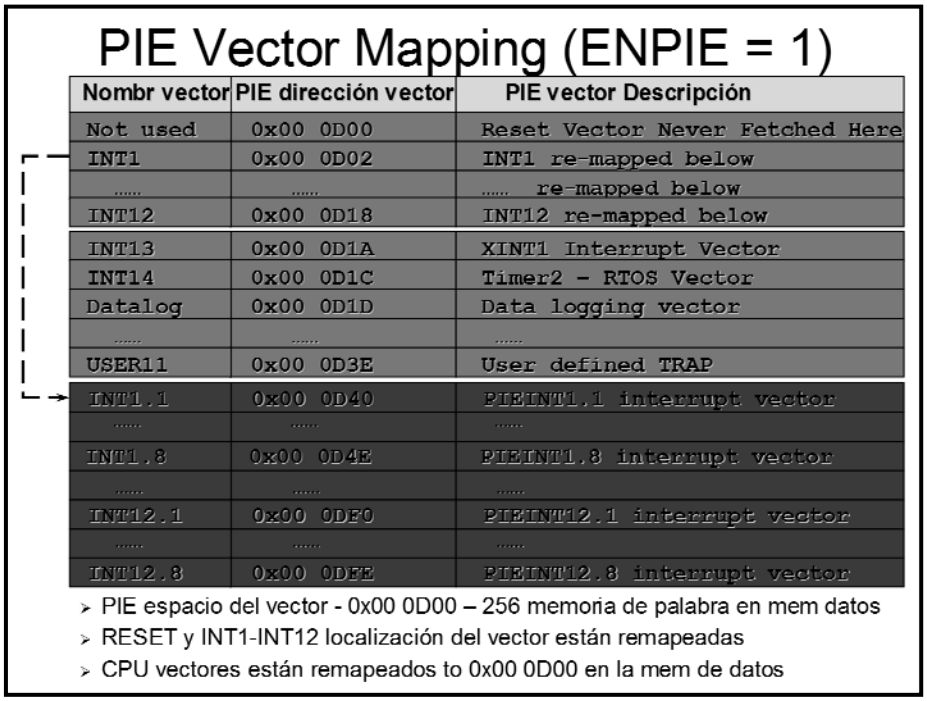


Tabla 5.15: Redireccionamiento de los vectores.

Como se puede ver en la siguiente figura, el área de direcciones desde 0x00 0D40 hasta 0x00 0DFF es usada como el área de expansión. Como podemos tener 32 bits para cada vector de interrupción individual desde PIEINT1.1 hasta PIEINT12.8.

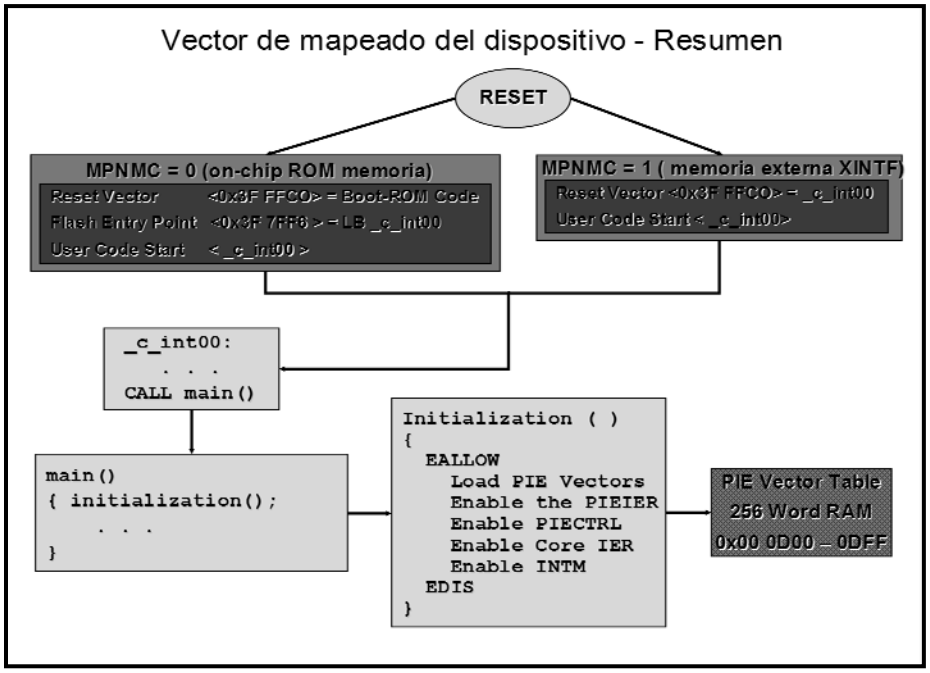


Figura 5.16: Vector del mapeado del dispositivo.

1.5.8.- Temporizadores de propósito general del C28x

El C28x tiene 3 contadores de 32 bits. El siguiente diagrama de bloques (Figura 5.17) representa la estructura de uno de los contadores:

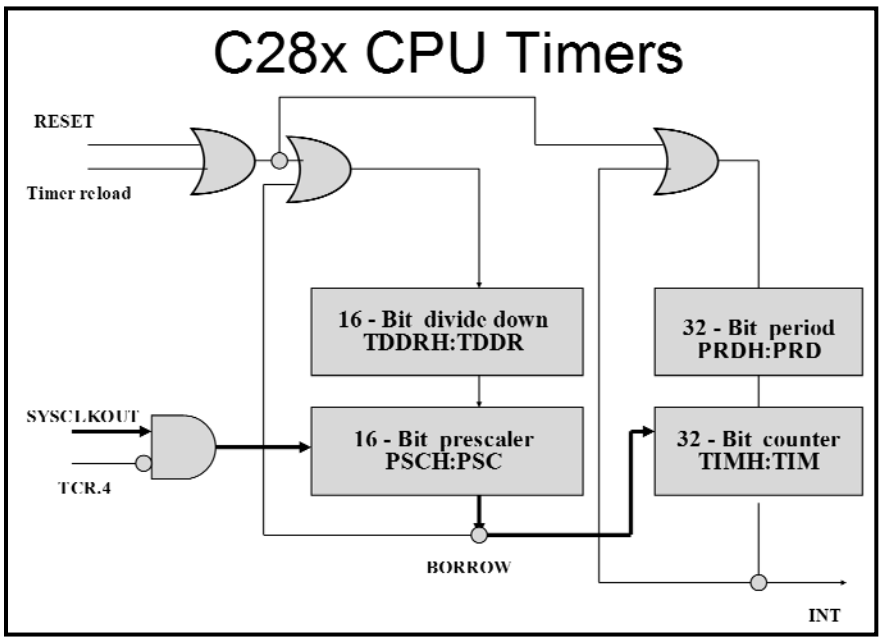


Figura 5.17: Temporizadores de propósito general del C28XX.

La señal de reloj que le llega al temporizador es "SYSCLKLOUT" que normalmente esta a 150 MHz, con un oscilador externo de 30MHz y una ratio de PLL de 10/2. Una vez que se ha seleccionado la fuente de reloj, se activa el contador con el bit 4 de registro TCR. El temporizador empieza a contar hacia atrás empleando un predivisor (PSCH: PSC) de 16 bits. La salida de este predivisor es la señal de reloj que ataca al contador de 32 bits (TIMH: TIM). Al final, cuando el contador llega a 0 se desborda y se genera una señal de interrupción que es transferida a la CPU (Figura 5.18).

El registro divisor de 16 bits (TDDRH: TDDR) es usado como un registro de recarga para el predivisor. Cada vez que el predivisor se desborda el valor desde el divisor baja un registro que es nuevamente cargado al predivisor. Una función similar de carga para el contador es usada por un registro periódico de 32 bits (PRDH_PRD) (Tabla 5.19).

De los tres temporizadores el 1 y el 2 son usados normalmente por Texas Instruments para operaciones en tiempo real con el “DSP/BIOS” mientras que el Timer 0 esta libre para uso general.

El uso del Temporizador 0 debe incluir el uso del PIE puesto que la interrupción que genera se produce a través de este gestor de interrupciones.

Por último indicar que cuando el DSP vuelve de un Reset los tres temporizadores están activados.

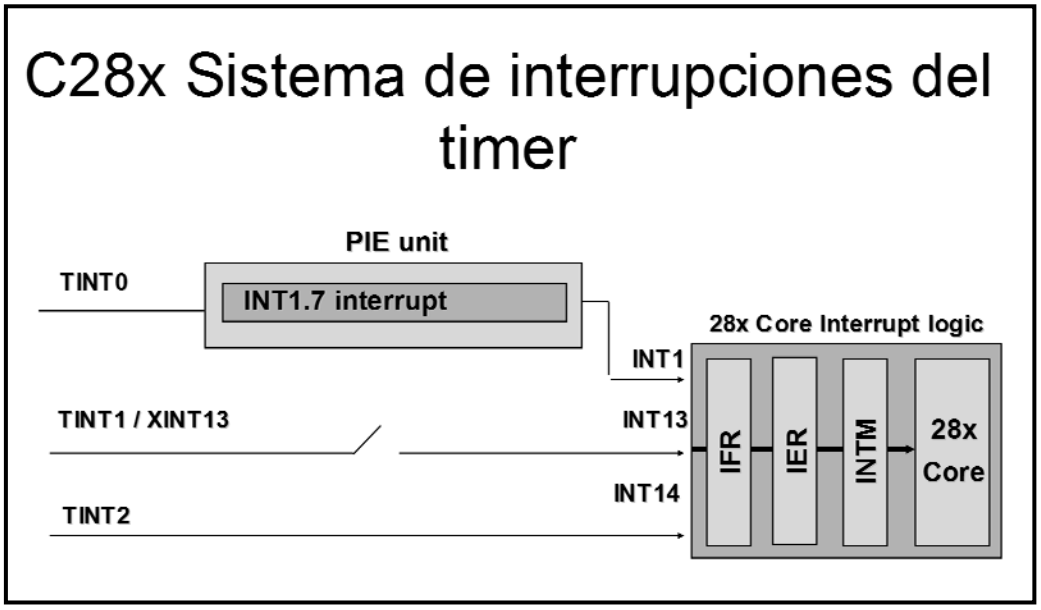


Figura 5.18: Interrupciones que producen los Timers de propósito general.

C28x Registros de timer

Address	Register	Name
0x0000 0C00	TIMER0TIM	Timer 0, Counter Register Low
0x0000 0C01	TIMER0TIMH	Timer 0, Counter Register High
0x0000 0C02	TIMER0PRD	Timer 0, Period Register Low
0x0000 0C03	TIMER0PRDH	Timer 0, Period Register High
0x0000 0C04	TIMER0TCR	Timer 0, Control Register
0x0000 0C06	TIMER0TPR	Timer 0, Prescaler Register
0x0000 0C07	TIMER0TPRH	Timer 0, Prescaler Register High
0x0000 0C08	TIMER1TIM	Timer 1, Counter Register Low
0x0000 0C09	TIMER1TIMH	Timer 1, Counter Register High
0x0000 0C0A	TIMER1PRD	Timer 1, Period Register Low
0x0000 0C0B	TIMER1PRDH	Timer 1, Period Register High
0x0000 0C0C	TIMER1TCR	Timer 1, Control Register
0x0000 0C0D	TIMER1TPR	Timer 1, Prescaler Register
0x0000 0C0F	TIMER1TPRH	Timer 1, Prescaler Register High
0x0000 0C10 to 0C17	Timer 2 Registers ; same layout as above	

Tabla 5.19: Resgistros del Timer.

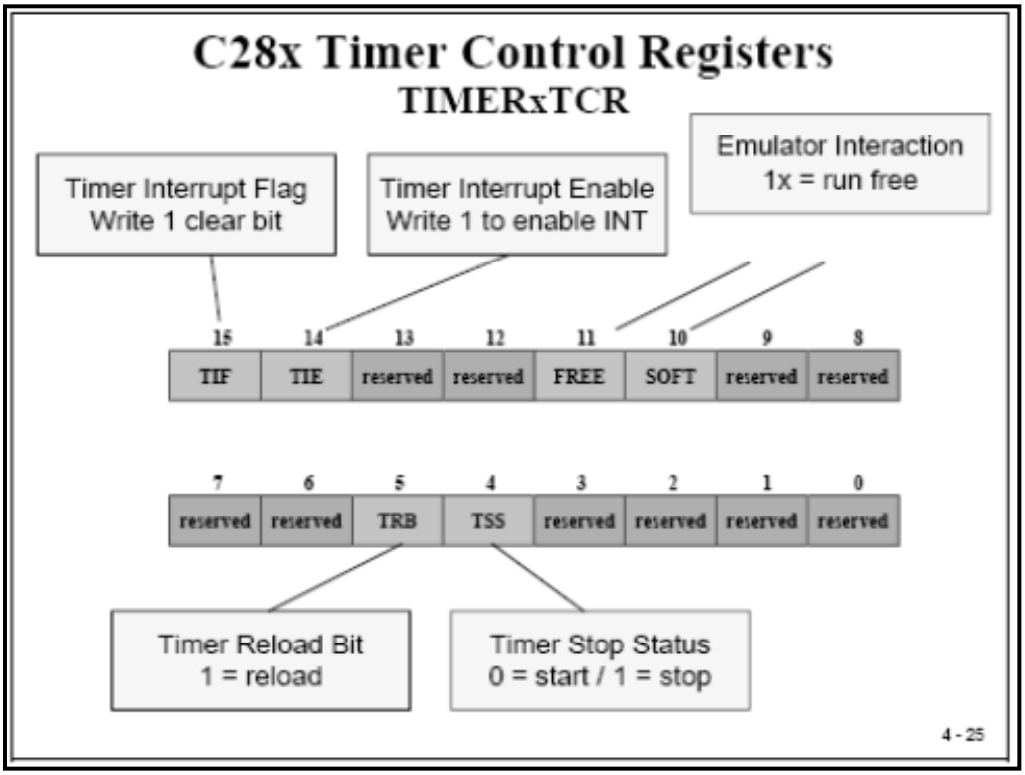


Figura 5.20: Registros de control del Timer.

1.6.- C28X EVENT MANAGER

1.6.1.- Introducción

El Event Manager (EV), o gestor de eventos del C28x es un periférico que procesa diferentes tipos de eventos temporales, generalmente asociados a uno o varios temporizadores. En concreto, el EV está dividido en dos bloques casi idénticos, el Event Manager A (EVA) y el Event Manager B (EVB). Cada uno de ellos posee dos temporizadores de 16 bits, denominados temporizadores T1 y T2 en el EVA y Temporizadores T3 y T4 en el EVB. También se les denomina contadores de tiempo, por lo que se usará indistintamente ambas denominaciones en este capítulo.

Estos temporizadores son básicamente el núcleo del EV al que se le añaden otros elementos como generadores de señales PWM y unidades de captura. Es muy importante no confundir estos temporizadores con los temporizadores de propósito general de 32 bits que posee el F2812. Estos temporizadores han sido detallados previamente en el capítulo 5.

La diferencia más importante entre los temporizadores del EV y los temporizadores del nucleo C28x, radica en el sistema de entradas/salidas y en el modo de funcionamiento, ya que pueden contar hacia delante o hacia atrás a criterio del programador.

.Un temporizador del EV puede producir señales de hardware directamente desde un evento interno, por lo que este periférico se suele emplear para generar señales basadas en eventos del hardware, por ejemplo cuando el temporizador llega a un valor de cuenta determinado, o al final del periodo, etc…

La señal generada es un pulso digital con una amplitud binaria (0,1) y con la ayuda de la lógica implementada en el EV es posible modificar la frecuencia y/o el ancho de pulso de estas señales de salida, dando lugar a señales de las formas más variadas De hecho, gracias a esta funcionalidad se pueden conseguir señales del tipo PWM (siglas de “Pulse Width Modulation) en las que el ancho del pulso a nivel alto/bajo puede variar durante el tiempo de ejecución.

Este tipo de señales PWM se suele emplear en diferentes aplicaciones como control digital de motores, control de convertidores de potencia, etc…. El F2812 es capaz de generar hasta 16 señales PWM.

El Event Manager incorpora una unidad de captura empleada para medidas de tiempo, entre dos pulsos o señales digitales, y también una unidad de medida de pulsos en cuadratura, para discriminar el sentido de giro de dicho eje.

1.6.2.- Diagrama de Bloques del Event Manager

Cada bloque, EVA o EVB (Figura 6.1), está controlado por su propio bloque lógico. Esta lógica permite generar señales de interrupción hacia el gestor de interrupciones PIE del C28x para soportar los modos de operación.

Dos señales externas de entrada 'TCLKINA' y 'TDIRA' son señales de control opcionales y son usadas en algunos modos de operación. También está preparada para generar una señal de arranque al convertidor ADC. Un gran número de microprocesadores comunes usa esta petición del servicio de interrupciones para hacer lo mismo, el C28x lo hace automáticamente.

Los temporizadores 1 y 2 son contadores de tiempo de 16 bits con sus propias señales de salida opcionales T1PWM/T1CMP y T2PWM/T2CMP. Esas señales pueden observarse a través de los pines del F2812 siempre que se configuren los puertos de entrada/salida para esta posibilidad.

Las unidades ce comparación 1, 2 y 3 se emplean para generar 6 PWM señales usando el temporizador 1 como contador de tiempo base lo que lo hace adecuado para un gran número de aplicaciones técnicas que requieren exactamente 6 señales de control, por ejemplo: un motor trifásico o un convertidor de potencia trifásico.

Tres unidades independientes de captura CAP1, 2 y 3 se usan para la velocidad y las estimaciones de tiempo. Un pulso de entrada en uno de las líneas de CAP nos daría un registro de tiempo desde cada GP Timer 1 o 2. Este registro de tiempo es proporcional al tiempo entres este evento y el anterior.

La unidad QEP redefine 3 líneas de entrada CAP1, 2 y 3 para ser usadas como sensores de pulsos exteriores (QEP1, 2) y pulsos de 0 (QEPI1) para un encoder incremental.

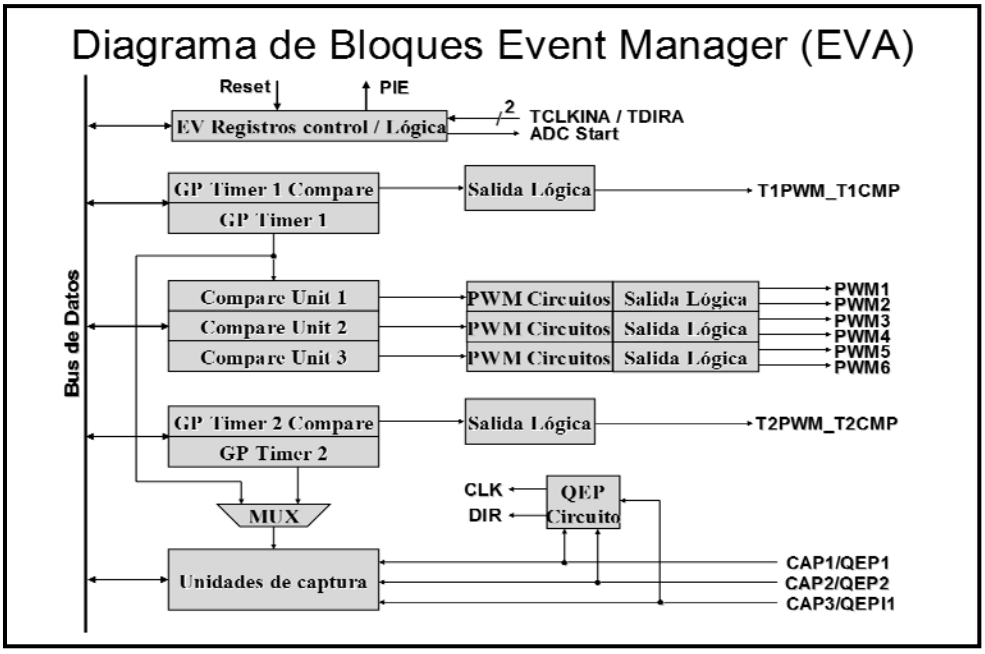


Figura 6.1: Diagrama de bloques del Event manager

1.6.3.- El temporizador de propósito general.

La lógica del temporizador de propósito general corresponde al bloque de comparación. Este subbloque está comparando continuamente el valor de un contador de 16 bits (TxCNT) con otros dos registros: Compare (TxCMPR) y Period (TxPR). Si hay una igualdad entre el *contador* y *compare* se manda una señal lógica a la salida para activar una señal externa (TxPWM). Si tienen un periodo diferente la señal se desactivará. Esta es el modo de funcionamiento denominado **modo asimétrico**. El otro modo de funcionamiento es el modo simétrico, que será explicado mas tarde. El registro GPTCONA controla el estado de las señales físicas de salida.

El reloj del temporizador puede seleccionarse de entre varias fuentes de entrada: una señal externa, la unidad QEP o una señal interna de reloj. La señal de reloj se elige con los bits 5 y 4 del registro TxCON, que controlan el multiplexor. En caso de seleccionar una señal de reloj interna se derivaría desde el predivisor de alta velocidad (HSPCLK), cuyo valor está indicado en el registro HSPCP.

Además hay que tener en cuenta otro predivisor adicionales (TPS, TxCON2-0), que tienen un factor de escala entre 1 y 128.

Indicar por último, que la dirección del contador también se puede seleccionar entre 3 modos que son:

- Modo continuo ascendente, cuenta desde cero hasta un valor.
- Modo continuo ascendente-descendente, cuenta desde cero a un valor y luego desciende hasta cero de nuevo.
- Modo de conteo y detención, cuenta y se detiene cuando llega a un valor.

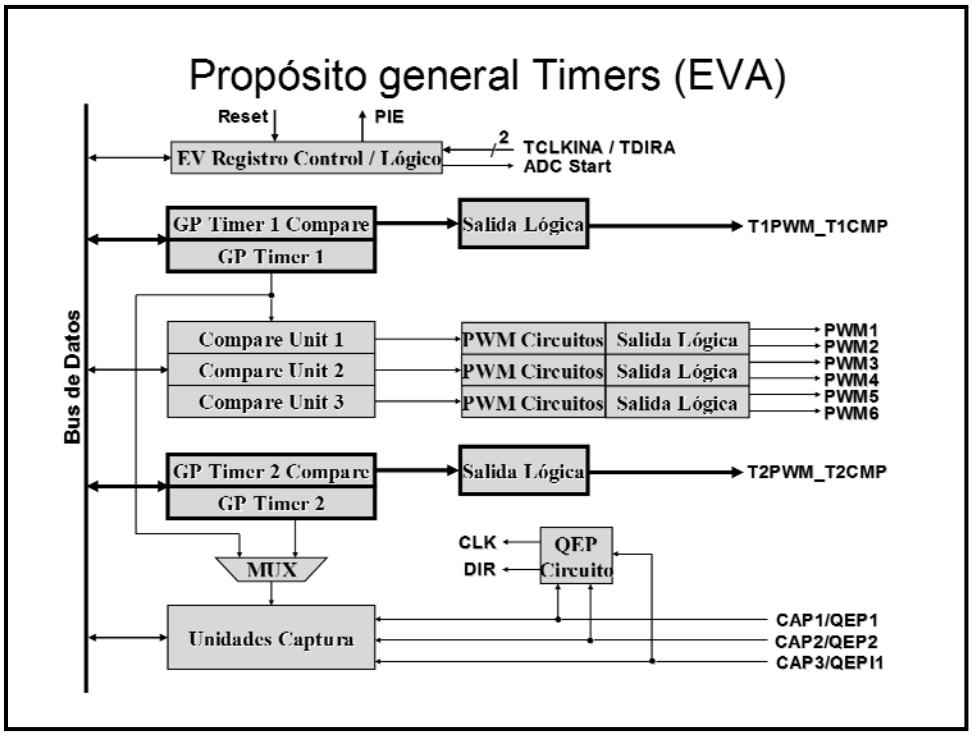


Figura 6.2: Timers de propósito general (Timer1 y Timer2)

Otra característica del EVA es el efecto de segundo plano (denominada “shadow” por el fabricante) implementada en determinados registros del temporizador.

Esto es porque en algunas aplicaciones se hace necesario modificar determinados valores de registros de comparación o de periodo dentro del periodo actual. Se consigue de esta forma preparar los siguientes valores de los registros modificados para que sean tenidos en cuenta en el siguiente periodo. Sin esta funcionalidad, habría que esperar al final de cada periodo y actualizar por software el valor, lo que acarrearía algunos retrasos que pueden llegar a ser críticos en determinadas aplicaciones.

1.6.4.- Modos de operación del contador de tiempo

El temporizador puede ser configurado para que cuente de varios modos, el primero de ellos es el modo ascendente continuo (Figura 6.3); este modo solo cuenta desde el 0 hacia un valor (periodo) y cuando llega a este punto empieza otra vez de 0. La apariencia de este modo es la de un diente de sierra.

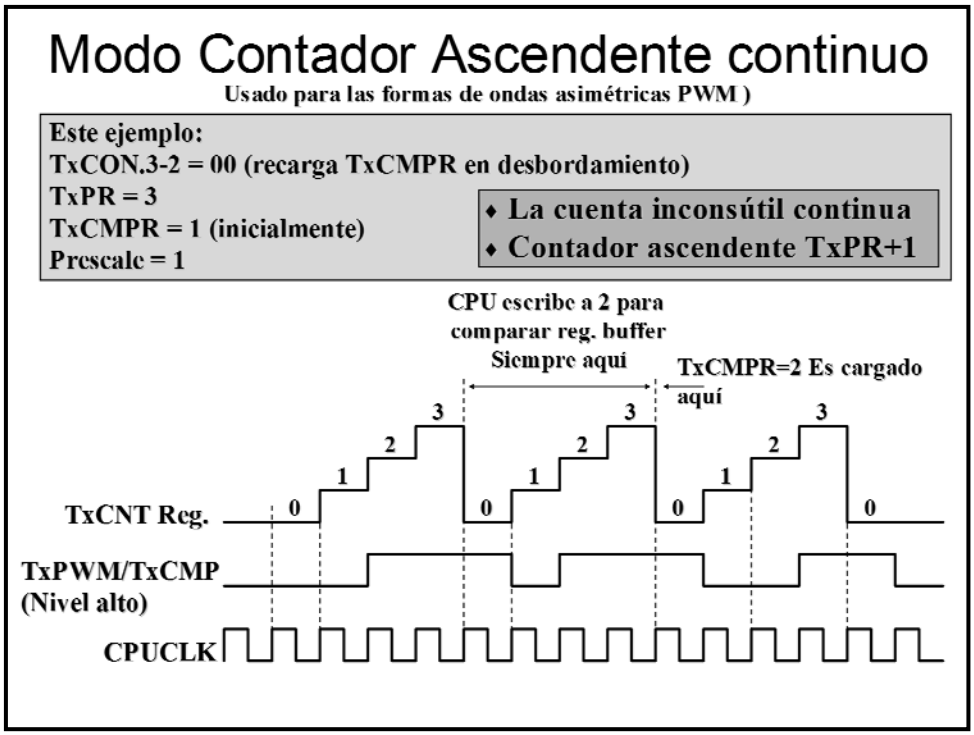


Figura 6.3: Modo de funcionamiento Continuous-Up (Diente de sierra).

El otro modo de funcionamiento es el modo continuo ascendente/descendente (Figura 6.4), este modo cuenta desde 0 hasta el valor del periodo y luego vuelve desde este valor hacia 0, esta señal se asemeja a un triangulo.

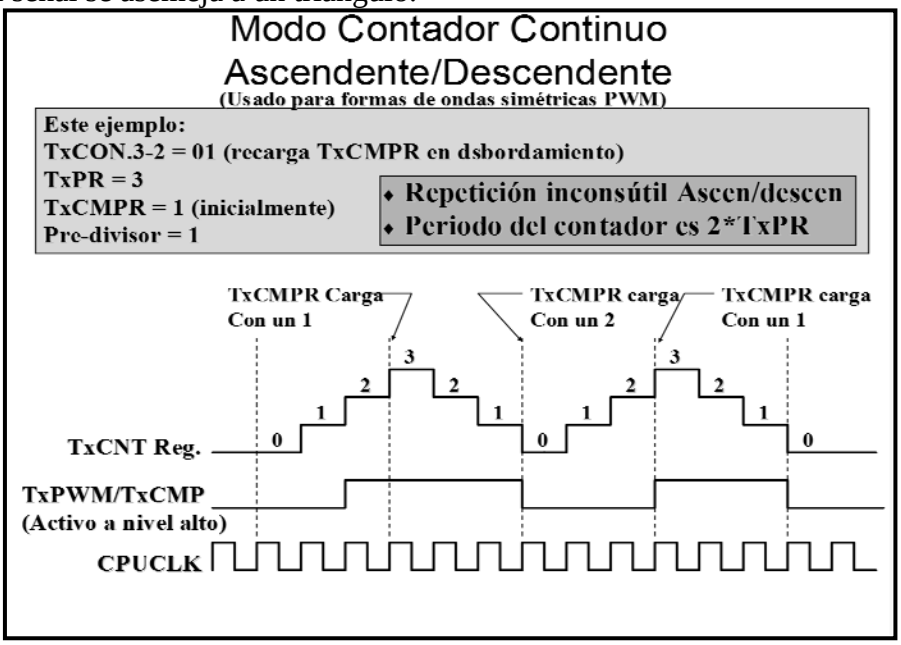


Figura 6.4: Modo de funcionamiento Continuous-Up/Down. (Triangulo)

1.6.5.- Fuentes de Interrupción

Cada uno de los contadores de tiempo del Event Manager A (EVA) puede generar 4 tipos de eventos que se traducen en la generación de sendas interrupciones, si la configuración del temporizador lo permite. Estos son:

- por desbordamiento del temporizador (el contador es igual a 0),
- por una comparación de tiempo (el contador es igual que el registro de comparación),
- por un periodo de tiempo (el contador es igual al registro de periodo), y
- por un desbordamiento del timer (el contador es igual a =0xFFFF, esta no viene reflejada en la figura siguiente.

La figura 6.5 nos muestra dos opciones para la forma de la señal de salida (TxPWM), activa a nivel alto y activa a nivel bajo en función del valor de un registro de control denominado GPTCONA. También podemos ver las posibles fuentes de interrupción.

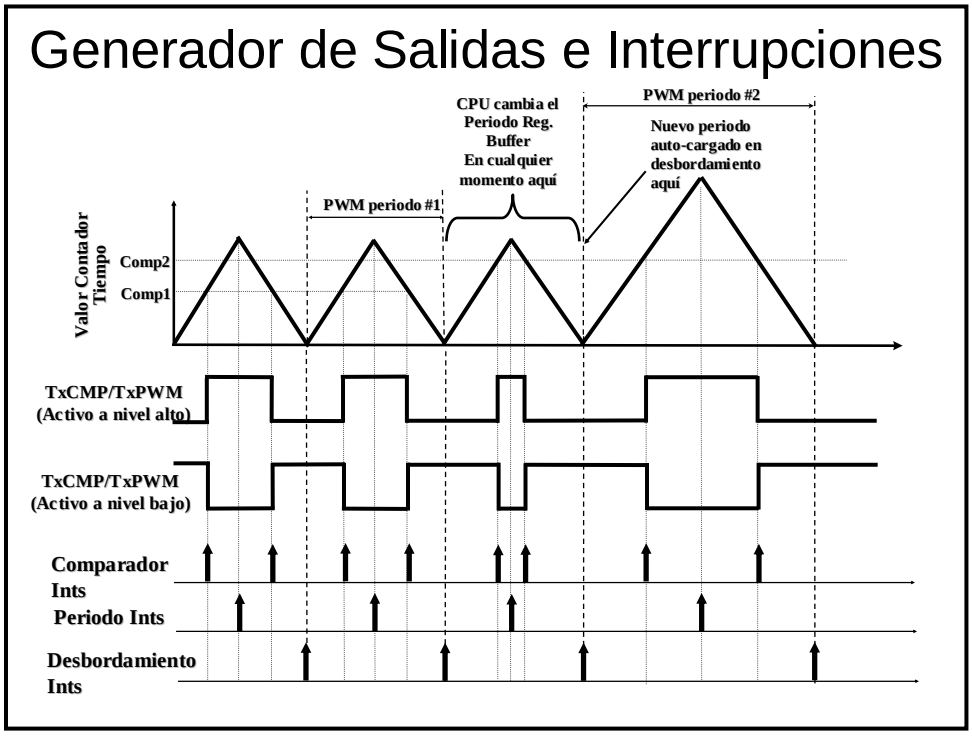


Figura 6.5: Generación de las posibles interrupciones del timer.

El ejemplo de la figura precedente asume un contador en modo up/down (ascendente/descendente) y que el timer empieza en el valor "Comp1" cargado dentro de TxCMPR y el periodo #1 en TxPR. En varios puntos en el periodo #2 el código cambia el valor en TxCMPR desde "Comp1" a "Comp2". Gracias al registro de comparación en segundo plano, este valor es metido dentro de la aplicación en la cual trabaja el usuario para la condición de recarga posterior. Así se consigue una nueva forma en las señales de salida para el periodo #3. En alguna parte del periodo #3 el código modificara el registro TxPR preparándolo para el periodo 4.

1.6.6.- Registros de control del Event Manager y el temporizador.

Para establecer el funcionamiento del EV y sus temporizadores es necesario configurar hasta cinco registros diferentes. Si además se quieren generar diferentes señales de interrupción habría que recurrir a algunos registros más. La siguiente figura enumera los registros con sus nombres abreviados y la dirección de memoria.

GP Registro de Tiempo

	Registro	Dirección	Descripción
EVA	GPTCONA	0x007400	Registro de control de propósito general A
	T1CNT	0x007401	Timer 1 Registro Contador
	T1CMPR	0x007402	Timer 1 Registro Comparador Buffer
	T1PR	0x007403	Timer 1 Registro Periodo Buffer
	T1CON	0x007404	Timer 1 Registro de Control
	T2CNT	0x007405	Timer 2 Registro de Contador
	T2CMPR	0x007406	Timer 2 Registro Comparador Buffer
	T2PR	0x007407	Timer 2 Regsitro de Periodo Buffer
	T2CON	0x007408	Timer 2 Registro de Control
EVB	GPTCONB	0x007500	Registro de Control de Propósito General Timer B
	T3CNT	0x007501	Timer 3 Registro Contador
	T3CMPR	0x007502	Timer 3 Registro Comparador Buffer
	T3PR	0x007503	Timer 3 Registro de Periodo Buffer
	T3CON	0x007504	Timer 3 Registro de Control
	T4CNT	0x007505	Timer 4 Registro Contador
	T4CMPR	0x007506	Timer 4 Registro Comparador Buffer
	T4PR	0x007507	Timer 4 Registro de Periodo Buffer
	T4CON	0x007508	Timer 4 Registro de Control

EXTCONA 0x007409 / EXTCONB 0x007509 ;Extensión del Registro de control

Figura 6.6: Registros de los temporizadores del EVA y EVB

1.6.7.- Registro de control del Temporizador GPTCONA.

La siguientes figuras reflejan la estructura del registro de control del temporizador y el significado de los campos que lo componen.

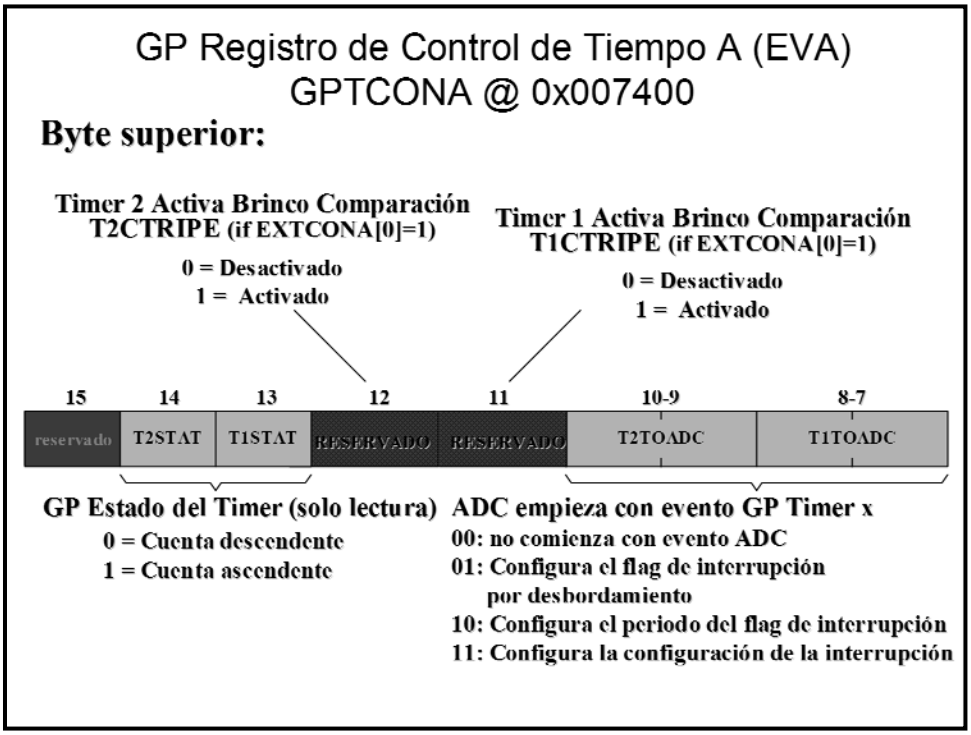


Figura 6.7: Registro GPTCONA (bits del 15 al 7)

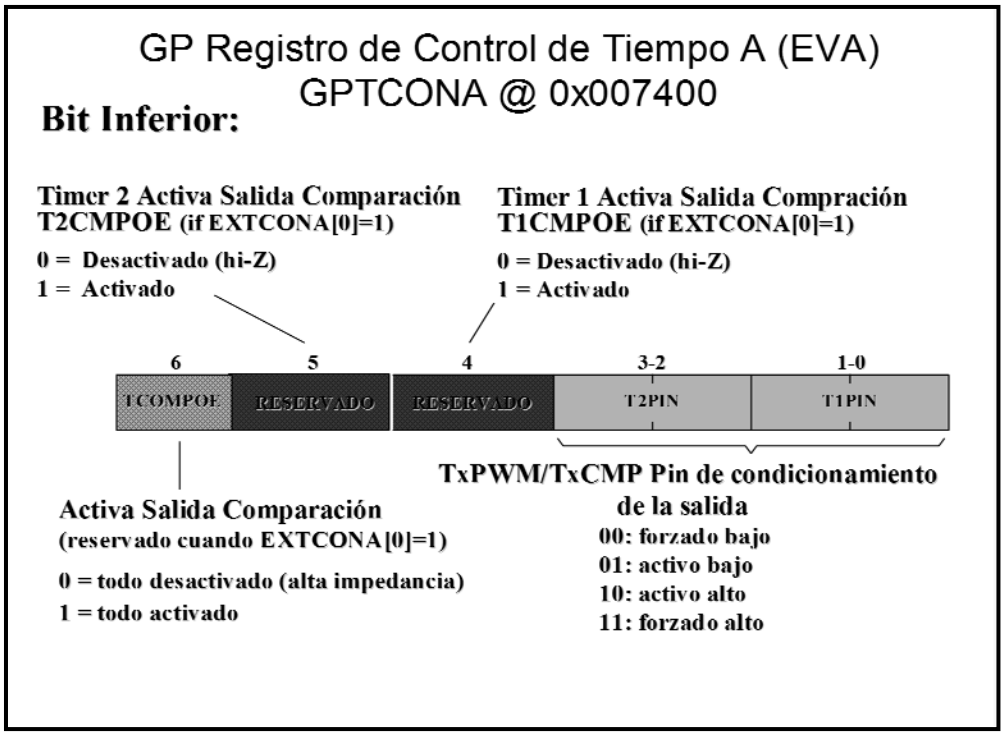


Figura 6.8: Registro GPTCONA (bits del 6 al 0)

Los bits 13 y 14 son bits de estado usados para informar de si el temporizador está contando hacia arriba o hacia abajo.

Los bits 11 y 12 están reservados para su uso conjunto con la unidad de captura. Para mayor información se recomienda ir al manual del fabricante

Los bits 10 hasta 7 son usados para arrancar automáticamente el ADC desde un evento específico.

El bit 6 es usado para activar dos señales físicas de salida para el temporizador 1 y el temporizador 2 simultáneamente. Aunque esto último sucede con el modo de operación incremental activado. Este extenso módulo es puesto en marcha configurando el bit 0 en el registro EXTCONA a 1. En este caso la definición de muchos de los bits de registro cambiará. El bit 6 no es muy usado; en lugar de eso los bits 4 y 5 se usan para activar o desactivar las señales de salida separadamente para el timer 1 y el timer 2.

. Los bits 3 a 0 definen la forma de la señal de salida.

1.6.8.- Registro del Control del Temporizador TxCON

A continuación configuraremos los registros de control del timer individualmente (Figura 6.9). La estructura de este registro esta mostrada en las siguientes dos figuras.

Los bits 14 y 15 son responsables de la interacción entre el compilador/depurador y la unidad de temporizador. Se emplean para definir cómo funcionará el temporizador en caso de un breakpoint por software. En el caso de estar conectado a un equipo y producirse una parada, podrían producirse daños en el equipo externo. Con estos dos bits es posible definir el estado del temporizador en este caso. En el caso de prácticas de laboratorio sería muy útil conseguir una parada inmediata cuando se para la ejecución del código.

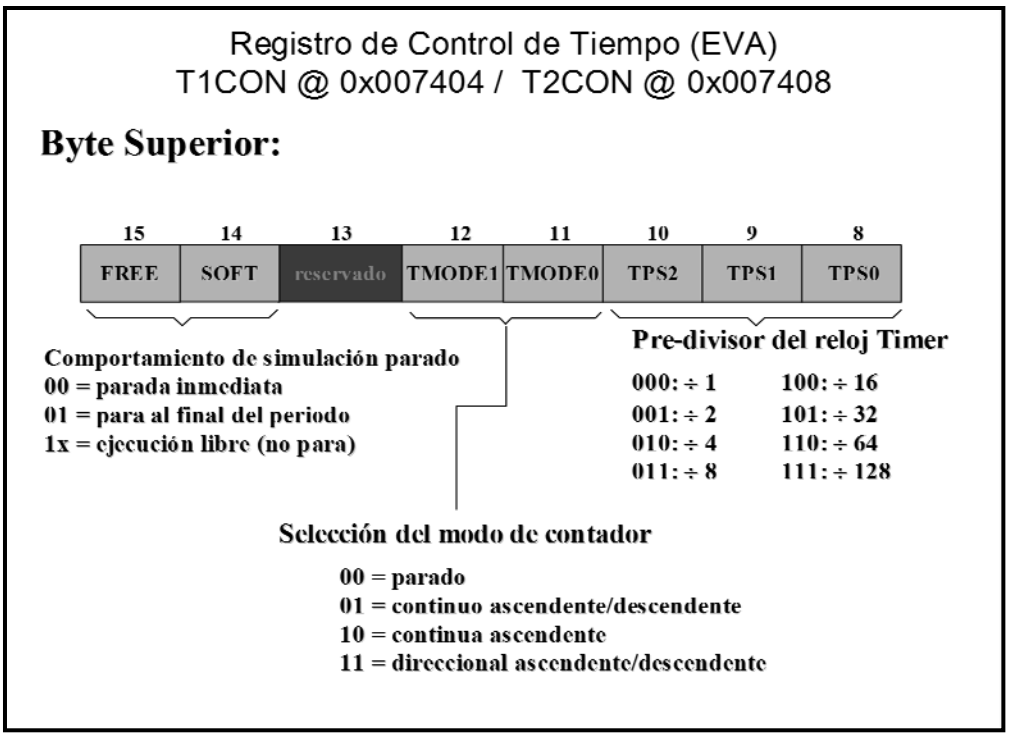


Figura 6.9: Registro T1CON (bits del 15 al 8)

Los bits 11 y 12 seleccionan el modo de operación. Se suelen emplear los dos modos intermedios, mientras que el primero y el último tienen aplicaciones muy específicas, como por ejemplo, contar los eventos de una señal externa, o el tiempo entre dos señales, etc…

Los bits del 8 al 10 corresponden a un predivisor para el reloj de temporizador y se emplean para reducir la señal del reloj del contador. Recuerde que la señal de reloj interna se

genera en el oscilador externo (30MHz), y pasa por un PLL interno (hasta 150MHz) y se puede dividir por dos factores, el predivisor HSPCP (que divide por 2) y el TPS (desde 1 hasta 128)

Por ejemplo, para configurar el temporizador y genere eventos con un periodo de 100 milisegundos, teniendo en cuenta el estado de los predivisores y el PLL, es necesario cargar el valor del registro TxPR con un valor de 58593. Se ha obtenido este valor de la siguiente forma:

Periodo de reloj del temporizador = (1/Freq. externa) * 1/PLL * HISPCP * Timer TPS

Pulso de señal de reloj del temporizador = (1/30 MHz) * 1/5 * 2 * 128= 1.7067µs

Valor de TxPR = 100ms/1.7067µs = 58593.

Esta tabla refleja los valores de TxPR que hay que cargar al temporizador en modo de funcionamiento up/down para que genere ondas de la frecuencia deseada.

Frecuencia	TxPR	Preescala	PLL
400Hz	23500	4	5
500Hz	18700	4	5
4KHz	9375	1	5
5KHz	7500	1	5
40KHz	937	1	5
50KHz	750	1	5

El bit 6 activa el temporizador. Al final del proceso de iniciación podemos configurar desde el bit 6 hasta el bit 1 de comienzo del timer.

Los bits 4 y 5 seleccionan la fuente de reloj.

Los bits 3 y 2 definen el momento de recargar el valor del registro de comparación a partir del valor en segundo plano.

El bit 1 se usa para activar la operación de comparación cuando es necesaria, ya que en otros casos solo se necesita generar señales periódicas, y no es necesario generar eventos de comparación.

Con la ayuda del bit 7 podemos forzar el comienzo del temporizador 1 y 2 simultáneamente. En este caso ambos temporizadores empezaran si el bit 6 (TENABLE) del registro T1CON esta a 1.

El bit 0 fuerza al temporizador 2 a usar el registro de periodo del temporizador 1 como una base para generar un periodo sincronizado para los dos temporizadores.

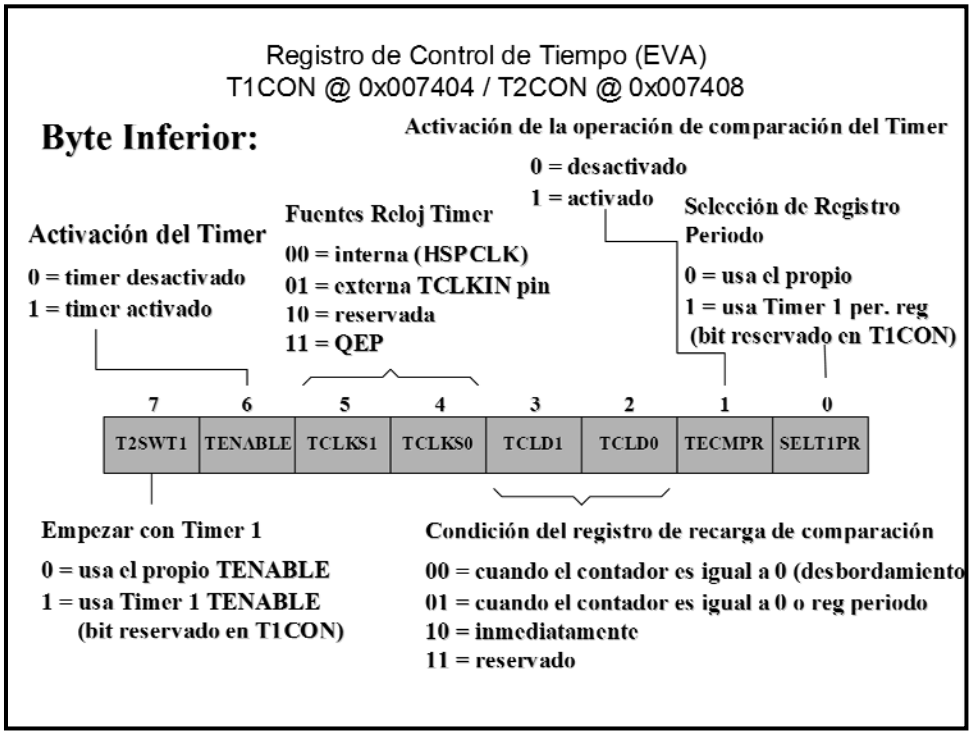


Figura 6.10: Registro T1CON (bits del 7 al 0)

1.6.9.- Interrupciones de los timer en el GP

El Event Manager indica diferentes eventos en función de la configuración. Estos eventos se pueden traducir en interrupciones que llegan al núcleo del C28x a través del gestor de interrupciones PIE.

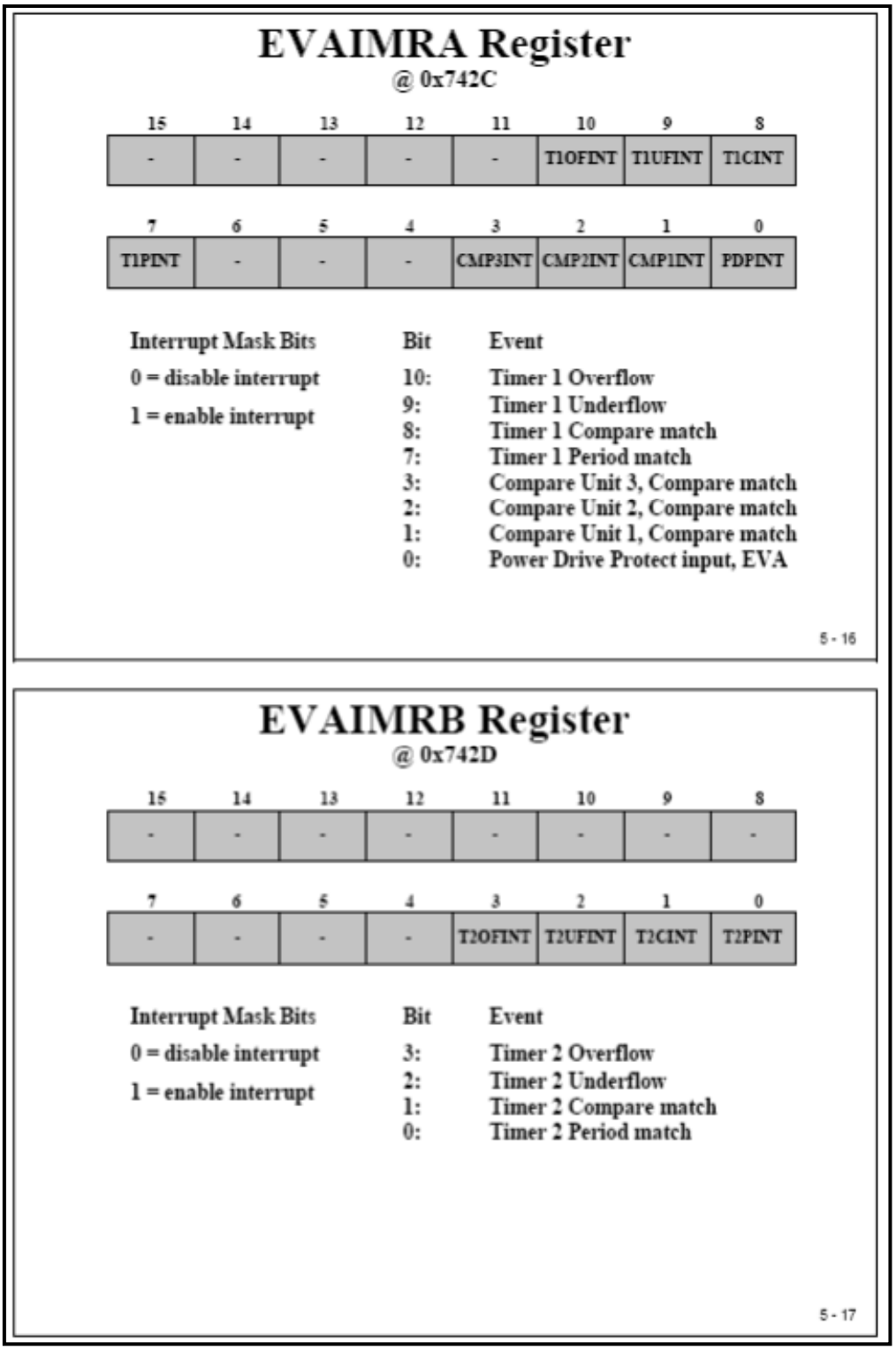
Las posibles fuentes de interrupción son las siguientes:

- Interrupción por fin de periodo
- Interrupción por comparación con alguno de los tres valores a comparar
- Interrupción por underflow.

Para hacer que se active cualquiera de las fuentes de interrupción del Event Manager se deben configurar los registros EVAIMRA, B o C según sea uno u otro temporizador. Cada bit de este registro corresponde a una interrupción.

Cuando se produce una interrupción, queda marcada en el registro de flags, EVAIFRA, B o C.

En las siguientes figuras se detalla la estructura de estos registros.



EVAIMRA Register
@ 0x742C

15	14	13	12	11	10	9	8
-	-	-	-	-	T1OFINT	T1UFINT	T1CINT

7	6	5	4	3	2	1	0
T1PINT	-	-	-	CMP3INT	CMP2INT	CMP1INT	PDPINT

Interrupt Mask Bits	Bit	Event
0 = disable interrupt	10:	Timer 1 Overflow
1 = enable interrupt	9:	Timer 1 Underflow
	8:	Timer 1 Compare match
	7:	Timer 1 Period match
	3:	Compare Unit 3, Compare match
	2:	Compare Unit 2, Compare match
	1:	Compare Unit 1, Compare match
	0:	Power Drive Protect input, EVA

5 - 16

EVAIMRB Register
@ 0x742D

15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-

7	6	5	4	3	2	1	0
-	-	-	-	T2OFINT	T2UFINT	T2CINT	T2PINT

Interrupt Mask Bits	Bit	Event
0 = disable interrupt	3:	Timer 2 Overflow
1 = enable interrupt	2:	Timer 2 Underflow
	1:	Timer 2 Compare match
	0:	Timer 2 Period match

5 - 17

Figura 6.11: Registro EVAIMRA y EVAIMRB (bits del 15 al 0)

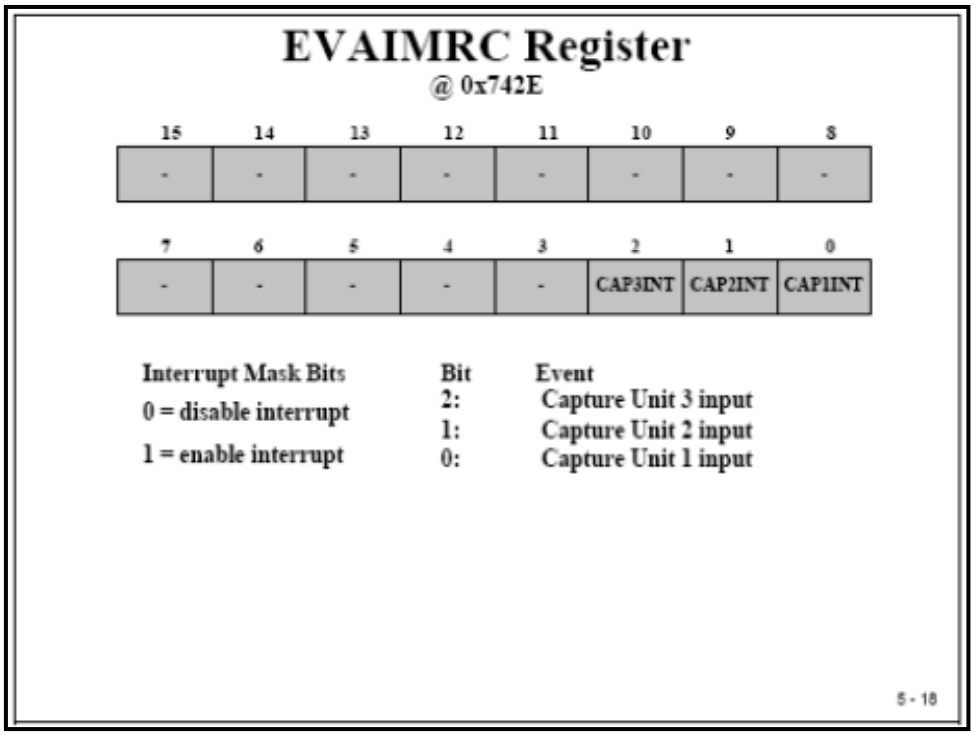


Figura 6.12: Registro EVAIMRC (bits del 15 al 0)

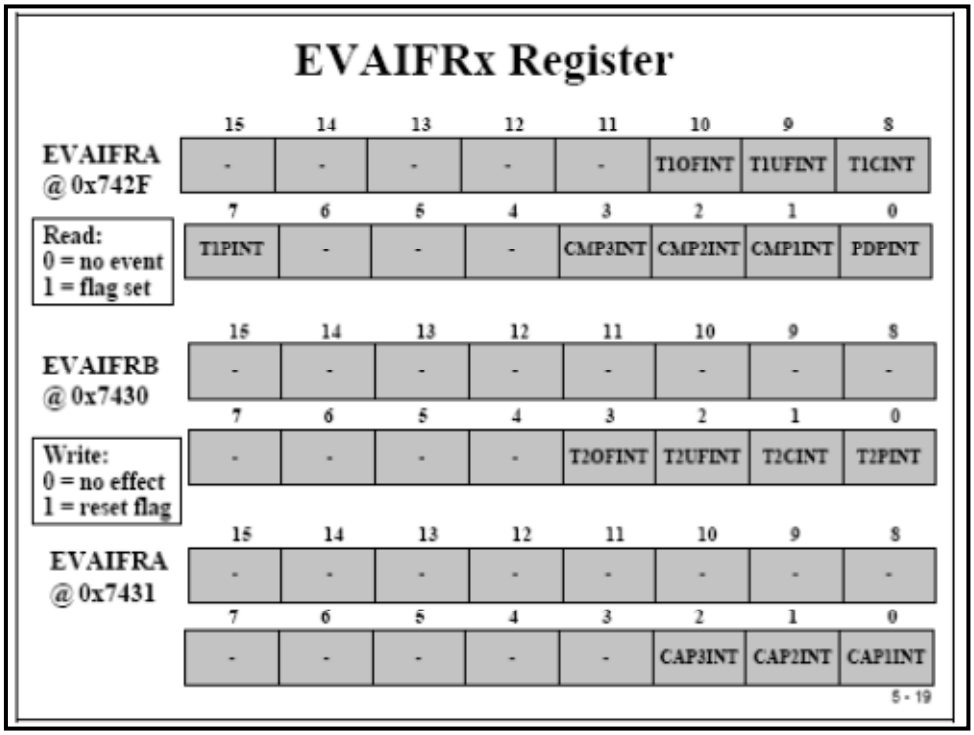


Figura 6.11: Registro EVAIFRx (bits del 15 al 0). Registra los flags de las interrupciones.

1.6.10.- Unidades de comparación del Event Manager

La unidad de comparación del Event Manager es una unidad que realiza la comparación del valor del temporizador con un valor determinado y permite generar, en función de la configuración de la lógica, hasta 6 señales PWM (Figura 6.12).

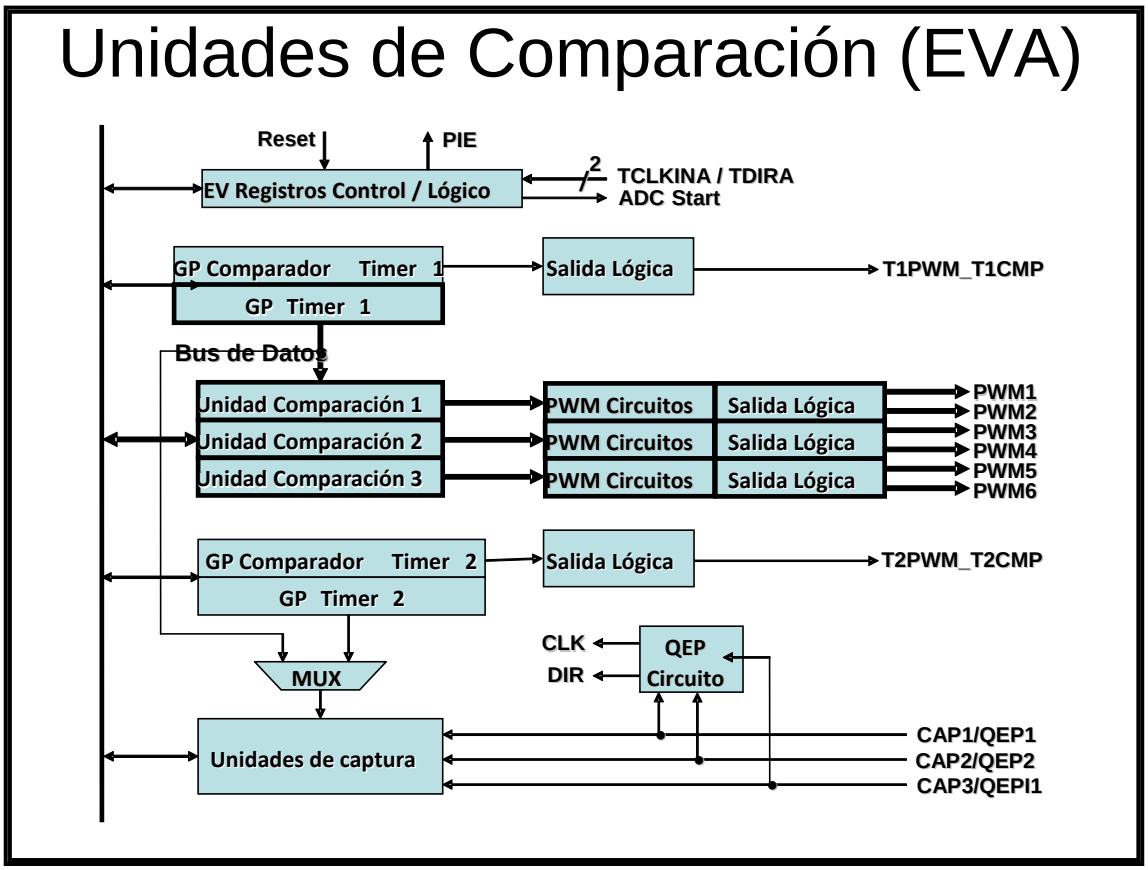


Figura 6.12: Unidades de comparación para las salidas PWM1 a la PWM6

La unidad de comparación del Event Manager es una unidad que realiza la comparación del valor del temporizador con un valor determinado y permite generar, en función de la configuración de la lógica, hasta 6 señales PWM.

Estas comparaciones pueden generar señales PWM, que en función del modo de funcionamiento del temporizador tienen una forma u otra, estas señales quedan reflejadas en las figuras 8.13 y 8.14

En ambas figuras se observa la generación de señales PWM, cada una con el contador de tiempo en un modo distinto, la que tiene el temporizador en modo up produce el primer cambio de flanco por comparación y el segundo por fin de periodo, mientras que la tiene el temporiazador en modo up/down genera los pulsos por alguna de sus tres posibles comparaciones.

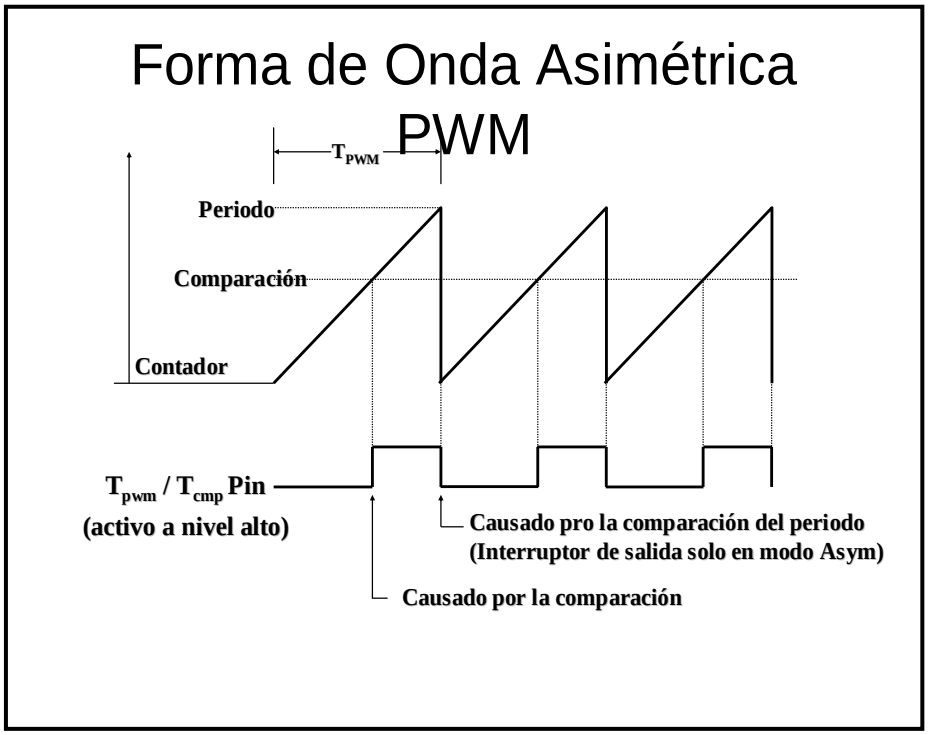


Figura 6.13: Forma de onda PWM con modo continuous-up

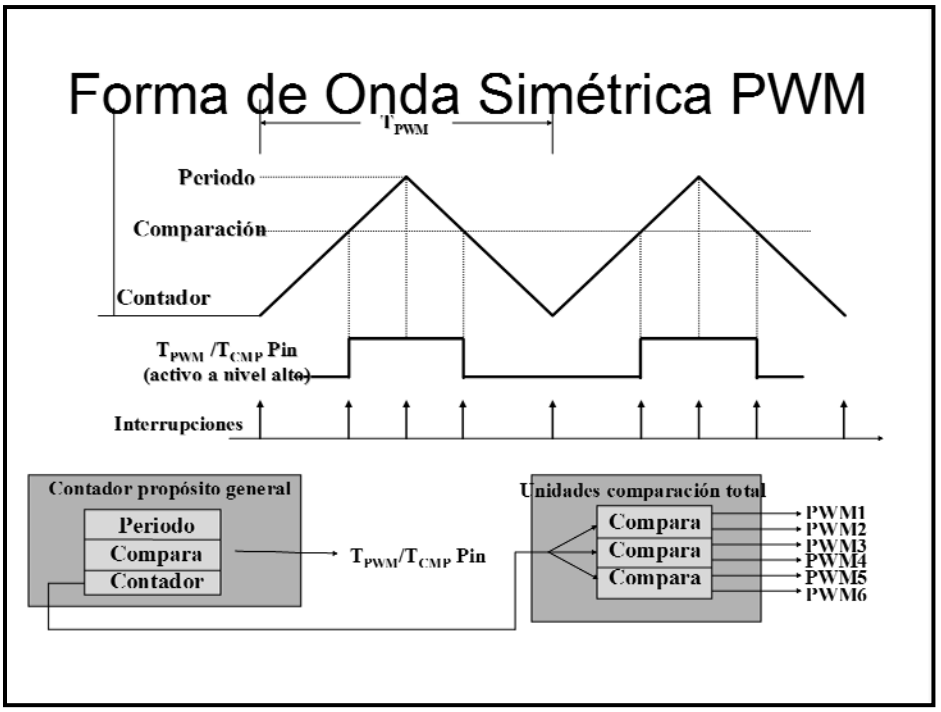


Figura 6.14: Forma de onda PWM con modo continuous-up/down.

El registro COMCONA nos configura las comparaciones de los timers y su estructura es la de las figuras 6.15 y 6.16.

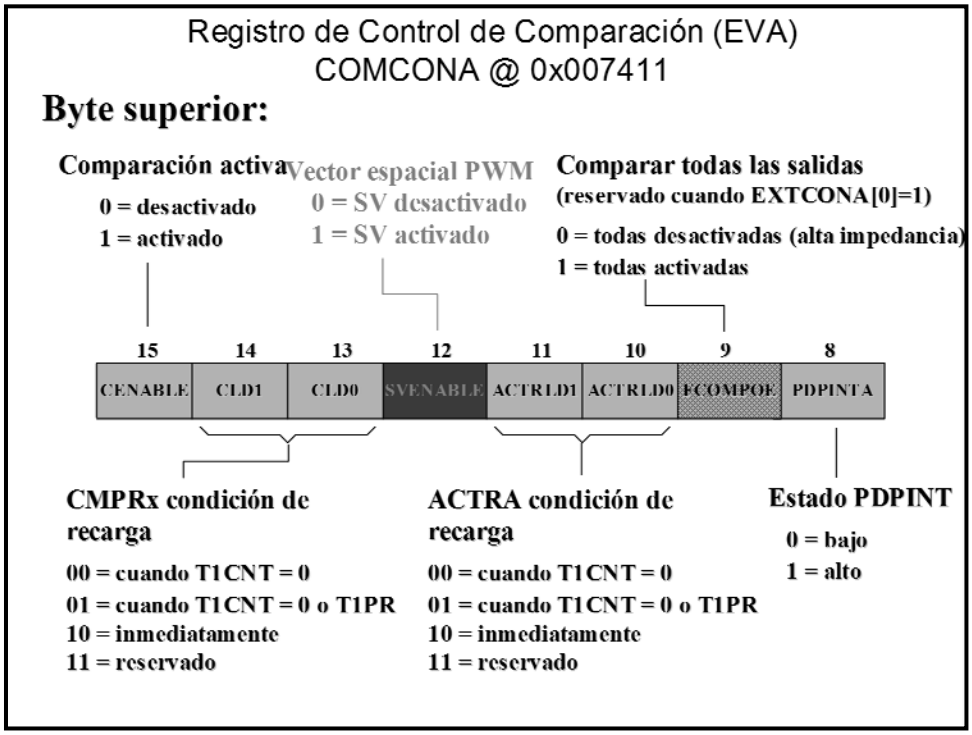


Figura 6.15: Registro COMCONA (bits del 15 al 8)

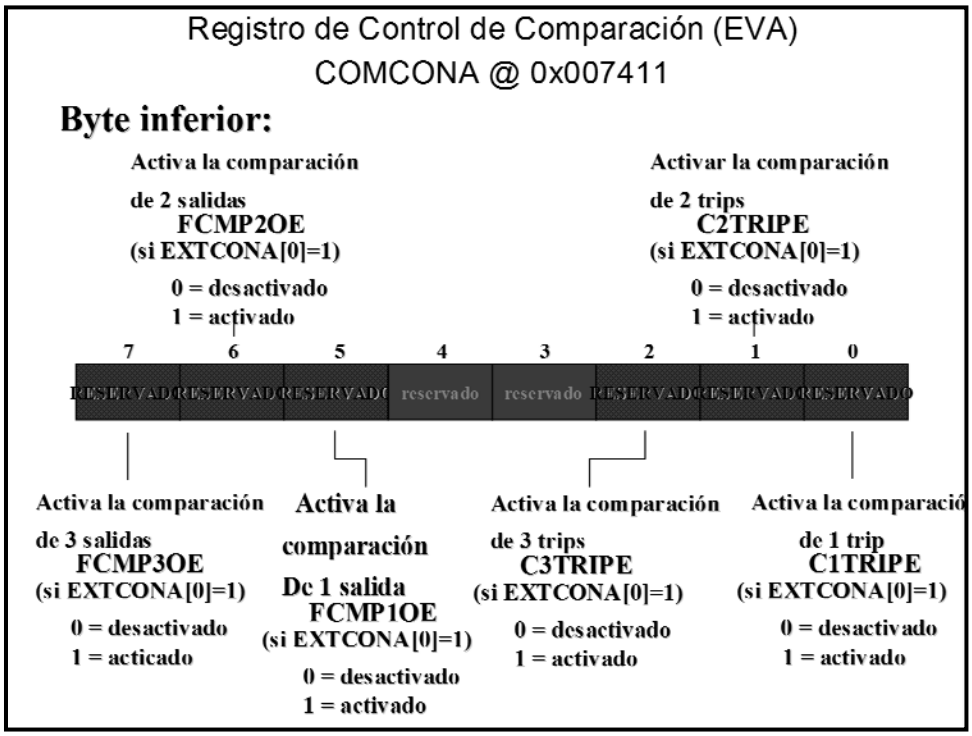


Figura 6.16: Registro COMCONA (bits del 7 al 0)

1.6.11.- Generación de tiempos muertos en las señales PWM.

La generación de tiempos muertos entre las señales evita problemas de conmutación, el tiempo muerto no es mas que un periodo de tiempo que existe entre que una onda cambia de nivel hasta que su inversa lo hace.

Para establecer el tiempo y las características de este tiempo muerto tenemos que recurrir a configurar el registro ACTRA y el registro DBTCONA, cuyas estructuras son las de las siguientes figuras 6.17 y 6.18.

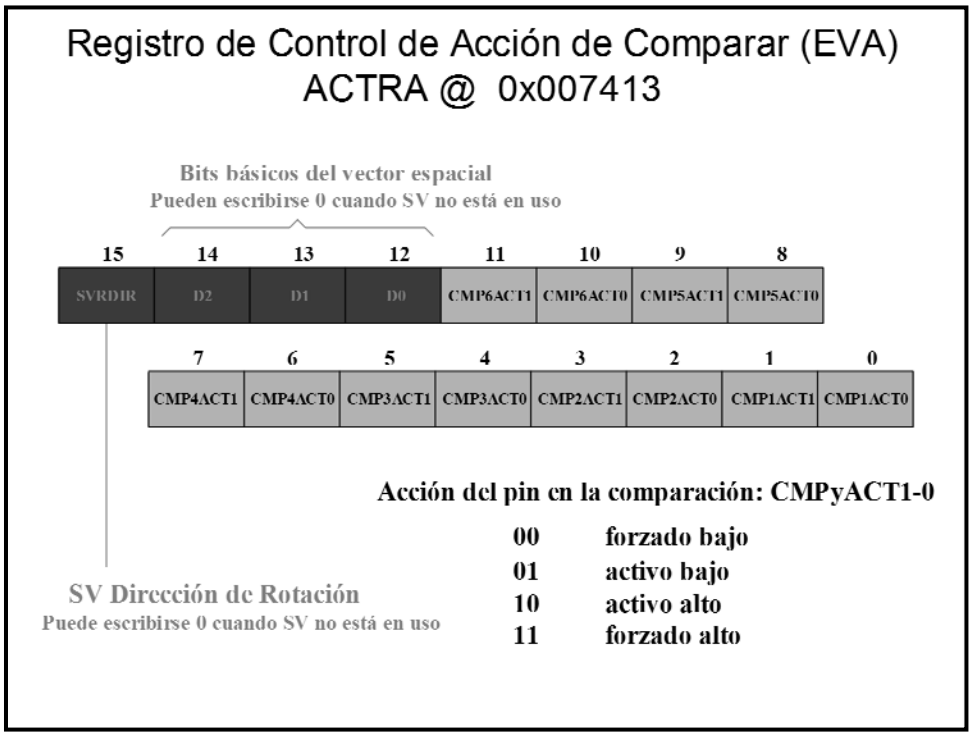


Figura 6.17: Registro ACTRA (bits del 15 al 0)

El registro ACTRA tiene agrupaciones de dos bits, cada uno de ellos controla a una pareja de señales PWM, forzando el cambio de flanco de una manera distinta, si lo fuerza a un nivel, cuando se produzca la comparación cambiará, mientras que si esta activo solo cambiará al nivel contrario.

El registro DBTCONA se utiliza para establecer cuanto tiempo habrá entre un cambio de flanco hasta el otro, los bits del 11 al 8 nos indican el tiempo a contar, del bit 7 al 5 si esta activado este tiempo muerto y del bit 4 al 2 si el tiempo a contar lleva preescala.

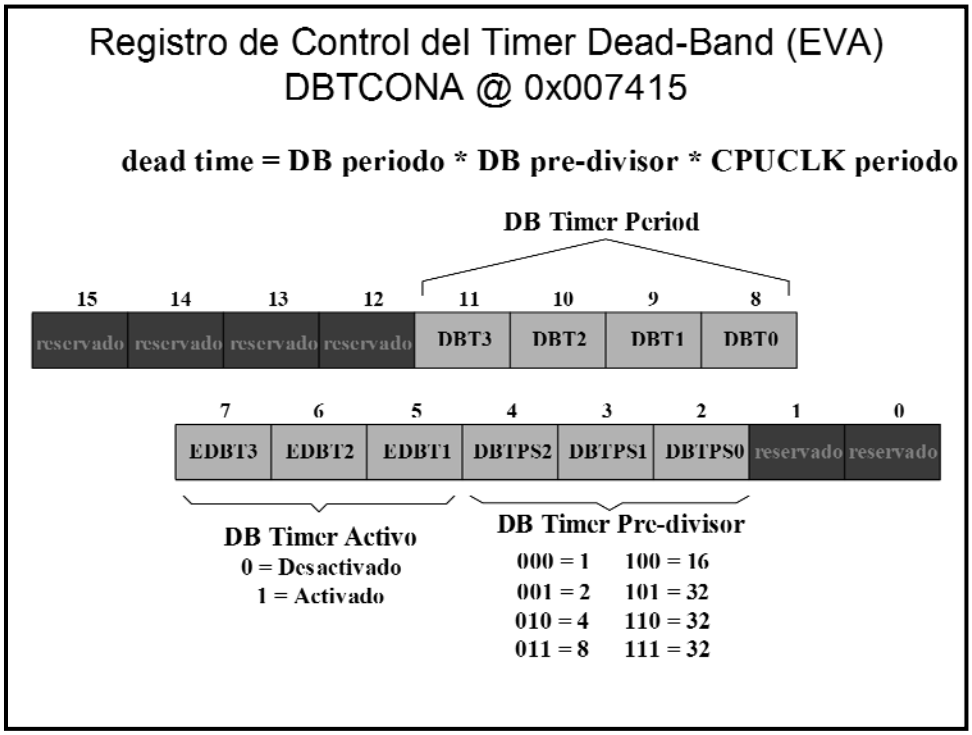


Figura 6.18: Registro DBTCONA (bits del 15 al 0)

En circuitos con dos o más interruptores, por lo tanto con dos o más señales de disparo, hay que tener precaución con el tiempo muerto que se elige, ya que si hay algún periodo de tiempo en el que dos interruptores de la misma rama conducen a la vez se pueden producir serios accidentes (Figura 6.19).

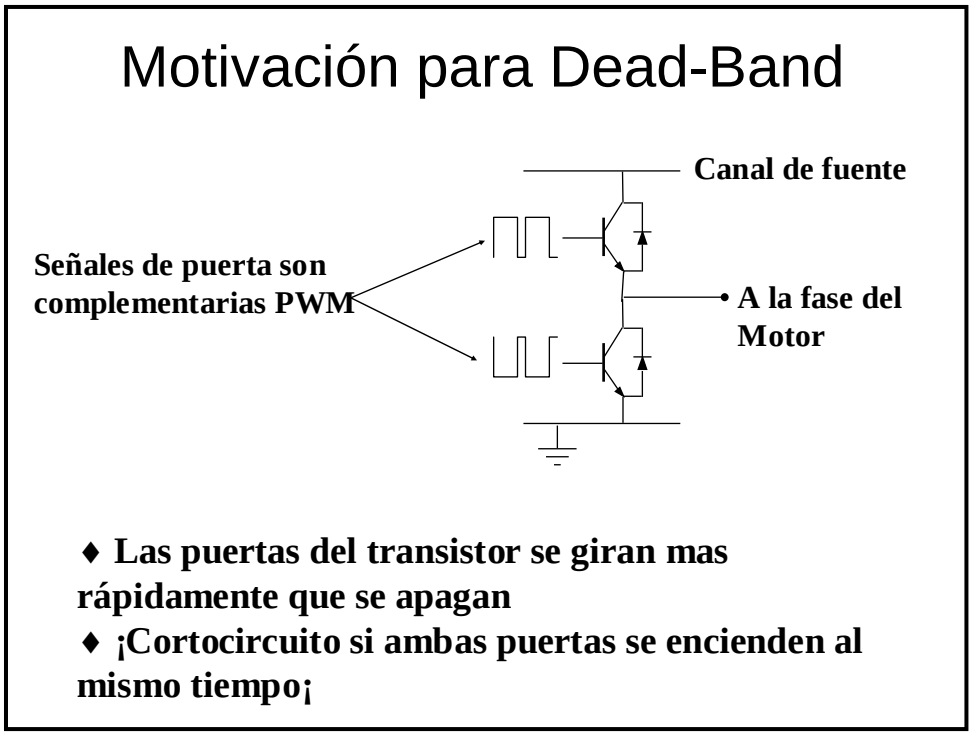


Figura 6.19: Motivación por la que se usa el tiempo muerto.

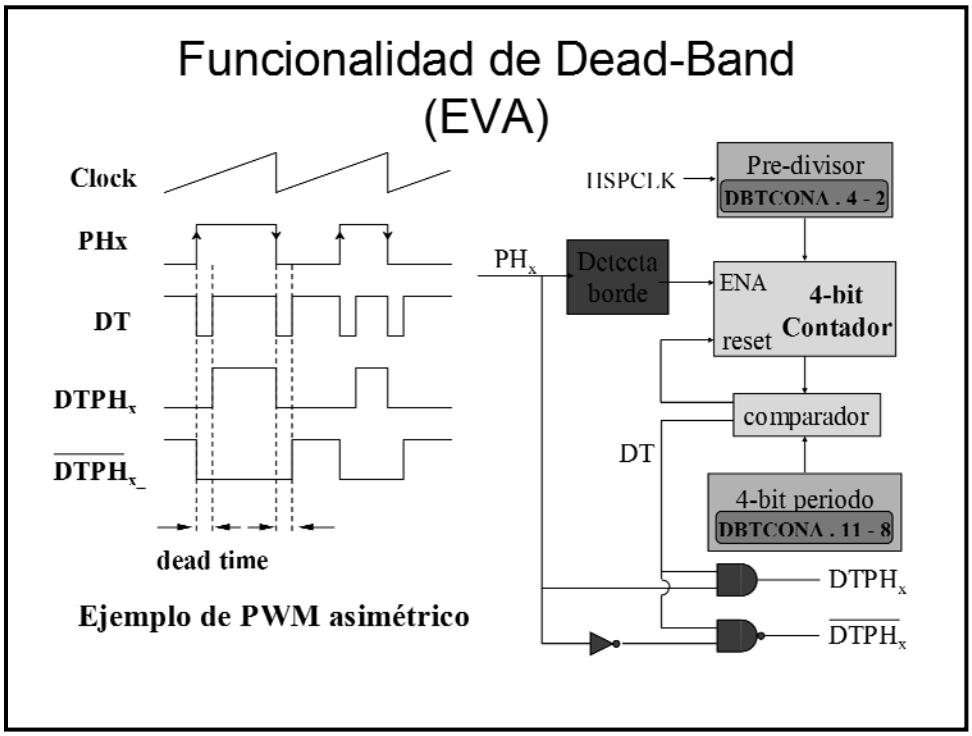


Figura 6.20: *Explicación de la funcionalidad del tiempo muerto.*

1.7.-Convertidor Analógico/Digital

1.7.1.-Introduccion.

Una de las unidades periféricas más importantes de un Microcontrolador es el convertidor analógico a digital (ADC).

Esta unidad proporciona un nexo importante entre el controlador y el mundo real.

La mayoría de las señales físicas tales como temperatura, humedad, presión, velocidad actual, aceleración son señales analógicas. Casi todos ellos se puede representar como un voltaje eléctrico entre Vmin y Vmax, por ejemplo, 0 ... 3V, que es proporcional a la señal original.

El propósito de el ADC es convertir este voltaje analógica en un número digital.

La relación entre el voltaje de entrada analógico (Vin), el número de dígitos binarios para representar el número digital (n) y el número digital (D) está dada por:

$$V_{in} = \frac{D * (V_{REF+} - V_{REF-})}{2^n - 1} + V_{REF-}$$

VREF + y VREF-son voltajes de referencia y se utiliza para limitar el rango de voltaje analógico. Cualquier entrada voltaje más allá de estos voltajes de referencia entregará un número digital saturada.

NOTA: Por supuesto todas las tensiones deben permanecer dentro de los límites de los valores máximos de acuerdo con la hoja de datos.

$$V_{in} = \frac{D * 3.0V}{4095}$$

La mayoría de las aplicaciones no sólo requieren una señal de entrada analógica a ser convertida en un valor digital; el lazo de control por lo general tiene varias señales diferentes de entrada del sensor. Por lo tanto, la C28x es equipado con 16 pines de entrada dedicado a medir los voltajes analógicos.

Estas señales son 16 multiplexado internamente, eso significa que se procesan de forma secuencial.

Para hacer la conversión, el ADC tiene que asegurarse de que durante el procedimiento de conversión no hay cambio del voltaje analógico de entrada Vin. De lo contrario el número digital sería totalmente erróneo. Un sample and hold (S&H)" interno se encarga de esto.

El C28x está equipado con dos unidades sample and hold, que se pueden utilizar en paralelo. Esto nos permite convertir dos señales de entrada (por ejemplo, dos corrientes) al mismo tiempo.

Pero hay más: El ADC C28x tiene un "auto-secuenciador" capacidad de 16 etapas.

Eso significa que el ADC automáticamente puede continuar con la conversión del canal de entrada siguiente después de que el canal anterior haya terminado.

1.7.2.- Descripción general del módulo ADC

Las características mas importantes de este módulo se detallan a continuación:

- Convertidor A/D de 12 bits.
- 16 entradas analógicas (rango 0-3V).
- 2 multiplexores que incluyen un circuito S&H. Cada multiplexor tiene 8 entradas analógicas.
- Modos de muestreo simultáneo y secuencial.
- Auto secuenciador:
    - Realiza de forma automática la conversión de 16 entradas en modo “cascada”.

    -En modo “dual” emplea 2 secuenciadores independientes de 8 entradas.

    - Hay un registro individual para la conversión de cada una de las entradas.
- Permite múltiples eventos de inicio de conversión:
    - Trigger externo, por software y por un timer del Event Manager.

Antes de entrar en los detalles de cómo programar el ADC interno vamos a resumir algunos detalles del módulo ADC.

La resolución digital del número convertido es de 12 bits. Asumiendo un rango de tensión de entrada de 0 ... 3 V se obtiene una resolución de voltaje de 3.0V/4095 =0.732mV por bit.

Tenemos dos unidades de S&H, que se pueden utilizar en paralelo ("muestreo simultáneo"). Cada unidad está conectada a 8 líneas de entrada multiplexadas. El secuenciador automático es programable y es capaz de convertir automáticamente hasta 16 señales de entrada.

El comienzo de una secuencia de conversión puede iniciarse a partir de cuatro fuentes:

• Mediante el software - acaba de establecer un bit de inicio al 1.
• Mediante una señal externa "ADCSOC".
• En un evento (período, comparar,desbordamiento) del Event Manager A .
• En un evento (período, comparar, underflow) del Event Manager B.

1.7.3.- ADC en modo cascada

En la siguiente figura se ha representado el diagrama de bloques para este modo de configuración del módulo ADC.

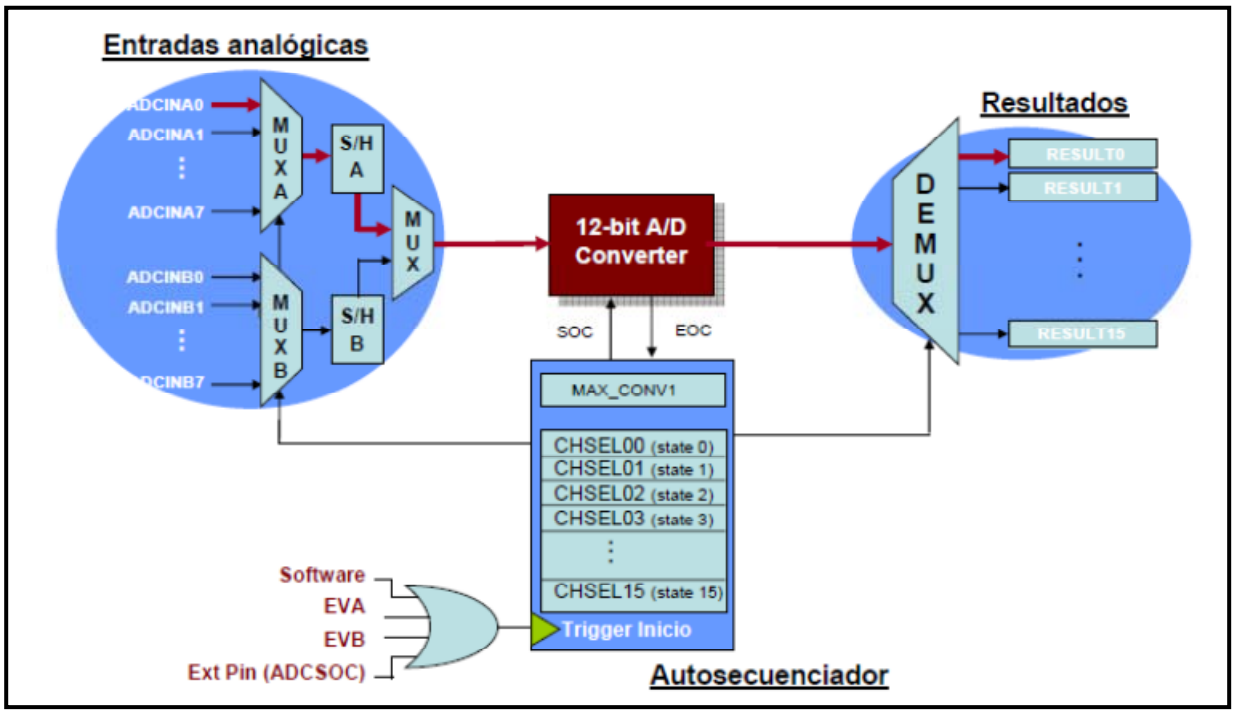


Figura7.1: *Diagrama de bloques para el funcionamiento de ADC en el modo "en cascada".*

Un Auto secuenciador controla el flujo de la conversión.

- Se debe establecer el número de conversiones (MAX_CONV1).
- Hay que indicar las líneas que son convertidas en cada etapa (CHSELxx).
- El resultado pasa a reg. Individuales (RESULT0 A RESULT15) para cada etapa.
- No es necesario utilizar rutinas de interrupción para cambiar de canal. El secuenciador lo
  hace de forma automática. (problema de jitter).

Podemos elegir entre dos opciones: toma de muestras "simultánea" o toma de muestras "secuencial".

En el modo simultaneo:

- Los 2 circuitos S&H se emplean en paralelo.
- 2 líneas de cada grupo se convierten a la misma vez en la misma etapa. Por ejemploADCINA3 y ADCINB3 en la etapa CHSEL00.

En el modo secuencial:

- Las líneas de entrada pueden ser conectadas a cualquier etapa del secuenciador.

Para activar una secuencia de conversión que puede utilizar un inicio de software mediante el establecimiento de un bit en particular, aunque también hay tres posibles opciones de inicio con eventos de hardware.

No hay necesidad de activar un servicio de interrupción (con su posible inestabilidad debido a las demoras de tiempo) para cambiar el canal de entrada, el secuenciador automático lo hará.

Se puede utilizar la interrupción de ADC al final de una secuencia para leer el bloque de registro de resultado.

1.7.4.- ADC en modo DUAL

En la siguiente figura se ha representado el diagrama de bloques para este modo de configuración del módulo ADC.

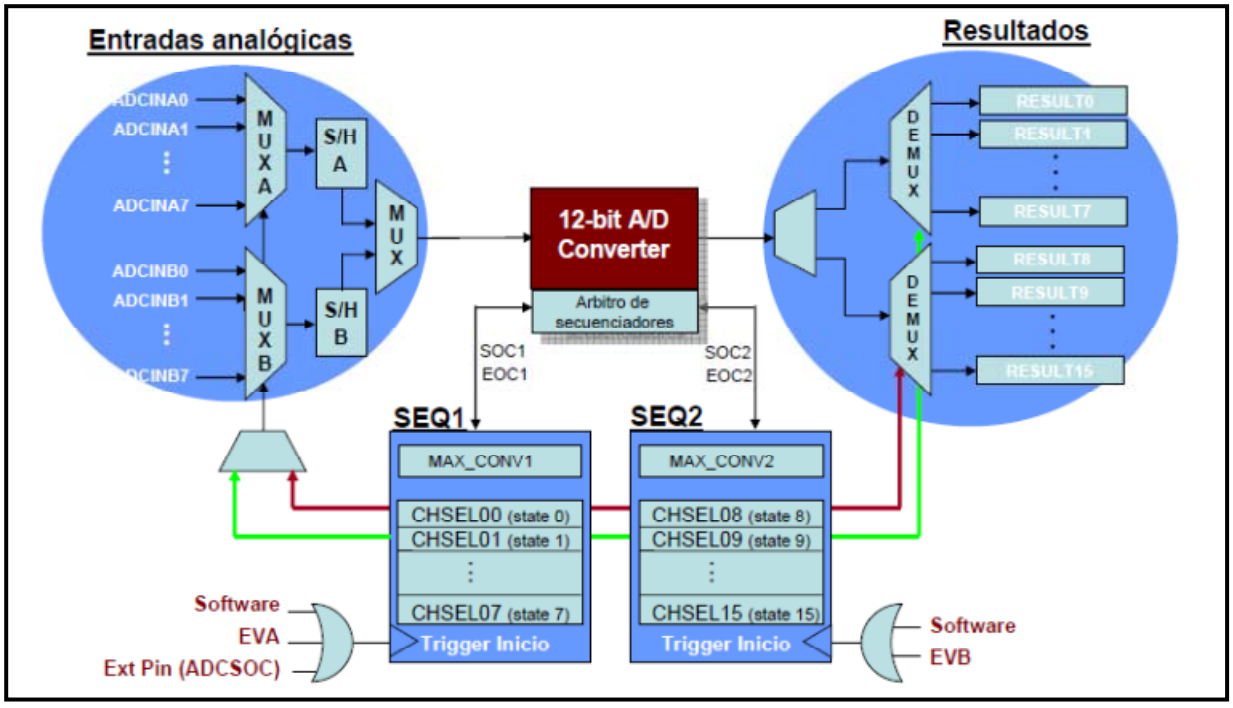


Figura 7.2: *Diagrama de bloques para el funcionamiento de ADC en el modo "dual"*

El segundo modo de funcionamiento de la ADC "Modo de secuenciador dual" divide el secuenciador automático en dos máquinas de estados independientes ("SEQ1" y "SEQ2").

Hay dos secuenciadores (SEQ1 y SEQ2).

- Se puede seleccionar cualquier línea de entrada en cualquiera de las 2x8 etapas.
- El SEQ1 controla RESULT0 a 7 y el SEQ2 controla RESULT8 a 15.

Este modo emplea EVA como trigger para SEQ1 y el EVB como trigger para SEQ2.

- De esta forma tenemos 2 A/D “virtuales” controlados por su propio HW.
- En caso de que se disparen los dos secuenciadores, el árbritro da prioridad al SEQ1.

1.7.5.-Reloj del módulo ADC

En la siguiente figura se ha representado el flujo de la señal de reloj que le llega al módulo ADC.

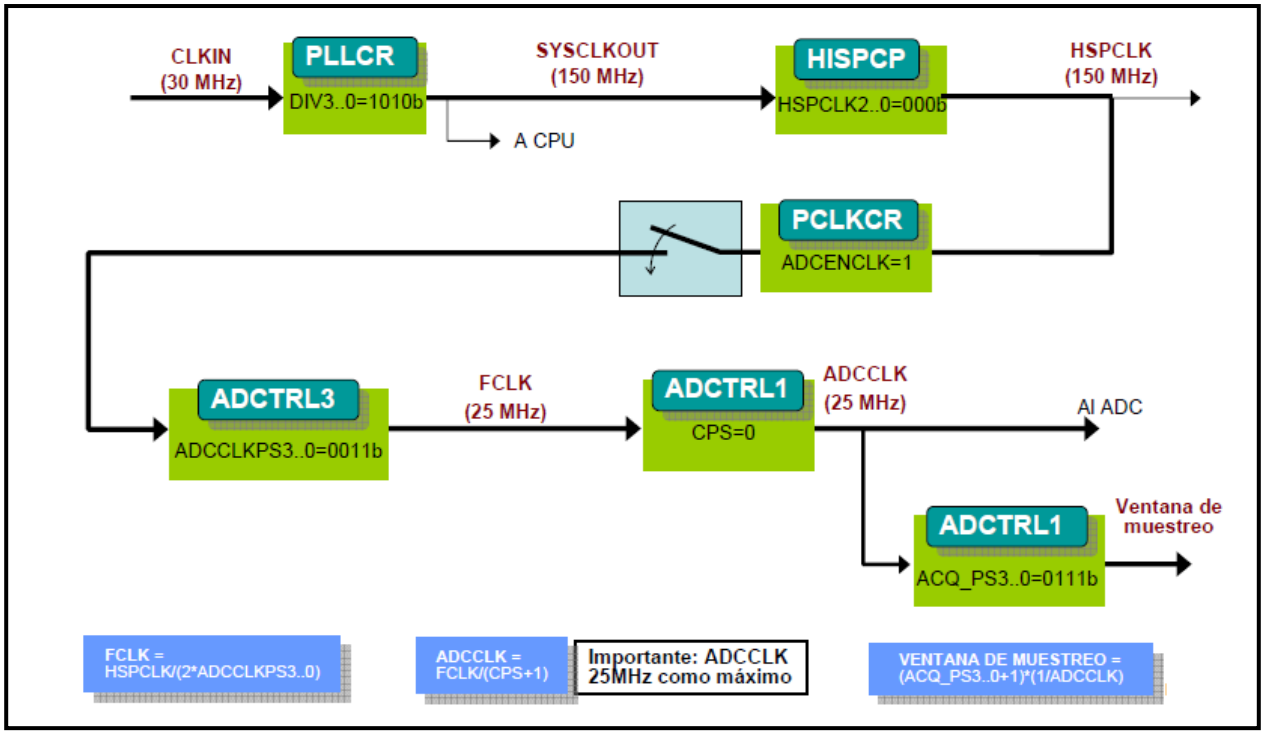


Figura 7.3: *Diagrama de bloques para el funcionamiento del reloj A/D*

Hay algunas limitaciones para la puesta en marcha del tiempo de conversión ADC. En primer lugar, el reloj básico fuente de la ADC es la HSPCLK reloj interno, no podemos utilizar cualquier velocidad de reloj. Este reloj se deriva del oscilador externo, multiplicado por PLLCR y dividido por HISPCP.

La segunda limitación es la frecuencia máxima para "FCLK" como la señal de entrada interna para la unidad ADC, esta señal se limita a 25MHz. Para ajustar este reloj tenemos que inicializar el bit de campo "ADCCLKPS" según corresponda. Bit "CPS" da la opción de otro divisor por 2.

El reloj "ADCCLK" es la base de tiempo para la canalización de procesamiento interno de la ADC.

Una tercera limitación es la ventana de muestreo controladas por el campo "ACQ_PS". Este grupo de bits define la longitud de la ventana que se usa entre el interruptor multiplexor y el momento que nos muestra la tensión de entrada (o "congelación") . Este tiempo depende de la impedancia de la línea de la entrada señal. Por lo tanto, es dependiente del hardware, no se puede especificar un período óptimo para todas las aplicaciones.

1.7.6.- Registros del módulo ADC

Tres registros de control "ADCTRL1 a 3" se utilizan para configurar los diversos modos de funcionamiento. La figura siguiente muetra el nombre del registro del ADC, su descripción y memoria.

Register	Address	Description
ADCTRL1	0x007100	ADC Control Register 1
ADCTRL2	0x007101	ADC Control Register 2
ADCMAXCONV	0x007102	ADC Maximum Conversion Channels Register
ADCCHSELSEQ1	0x007103	ADC Channel Select Sequencing Control Register 1
ADCCHSELSEQ2	0x007104	ADC Channel Select Sequencing Control Register 2
ADCCHSELSEQ3	0x007105	ADC Channel Select Sequencing Control Register 3
ADCCHSELSEQ4	0x007106	ADC Channel Select Sequencing Control Register 4
ADCASEQSR	0x007107	ADC Auto sequence Status Register
ADCRESULT0	0x007108	ADC Conversion Result Buffer Register 0
ADCRESULT1	0x007109	ADC Conversion Result Buffer Register 1
ADCRESULT2	0x00710A	ADC Conversion Result Buffer Register 2
ADCRESULT14	0x007116	ADC Conversion Result Buffer Register 14
ADCRESULT15	0x007117	ADC Conversion Result Buffer Register 15
ADCTRL3	0x007118	ADC Control Register 3
ADCST	0x007119	ADC Status and Flag Register

Figura7.4: *Registros del A/D*

REGISTRO ADCTRL1

Bit 14 ("RESET") se puede utilizar para restablecer la unidad toda ADC a su estado inicial. Siempre es bueno en la práctica aplicar un comando RESET antes de inicializar el ADC.

Los bits 13 y 12 definen la interacción entre el ADC y un comando emulador, similar a la comportamiento que vimos en el módulo gestor de eventos.

Los siguientes 4 bits definir la longitud de la ventana de muestra.

"CPS" se utiliza para dividir la frecuencia de entrada por 1 o 2.

Bit 6 ("CONT_RUN") define si el secuenciador automático comienza al final de una secuencia (= 0) y espera a otro disparador o si la secuencia debe empezar de nuevo inmediatamente (= 1).

Bit 5 ("SEQ1_OVRD") define dos opciones diferentes para el modo continuo. No vamos a utilizar este el modo en nuestras aplicaciones.

Por último bit 4 define el modo de secuenciador para ser una máquina de estado de 16 (= 1) o para funcionar como dos máquinas independientes del Estado de 8 estados.

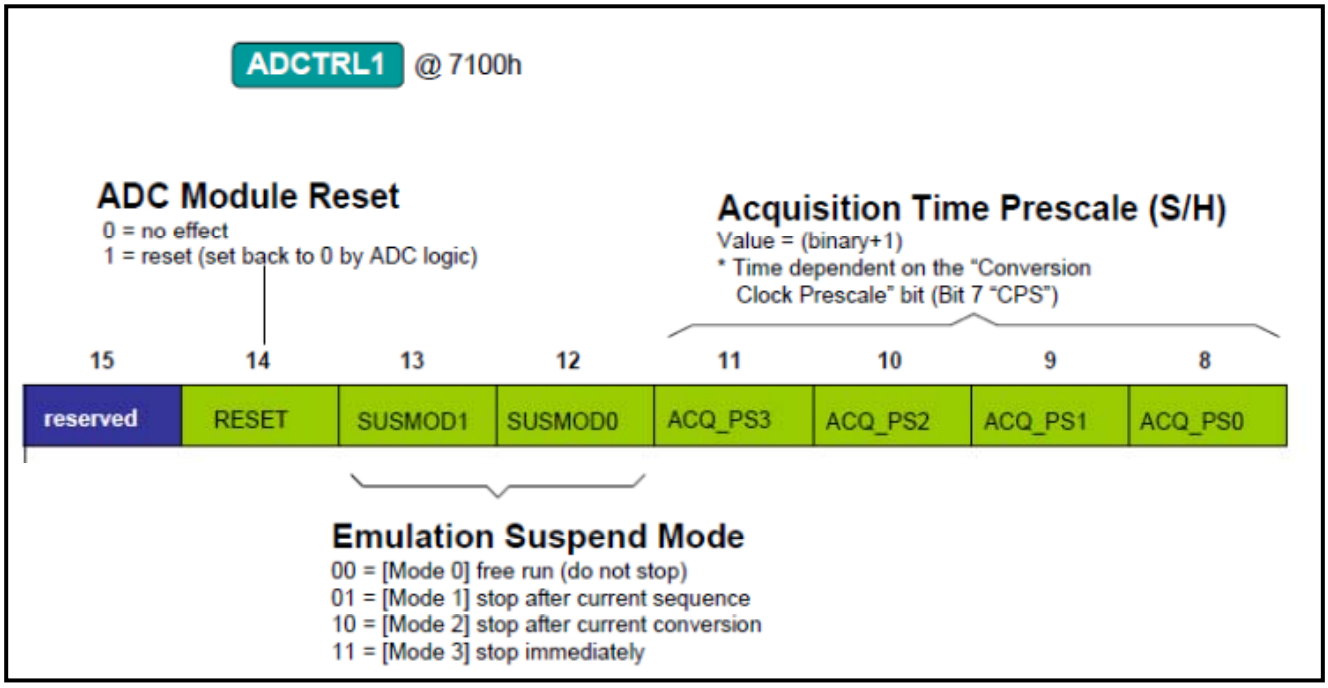


Figura 7.5: *Registro de control ADCTRL1. Bits 8 a 15*

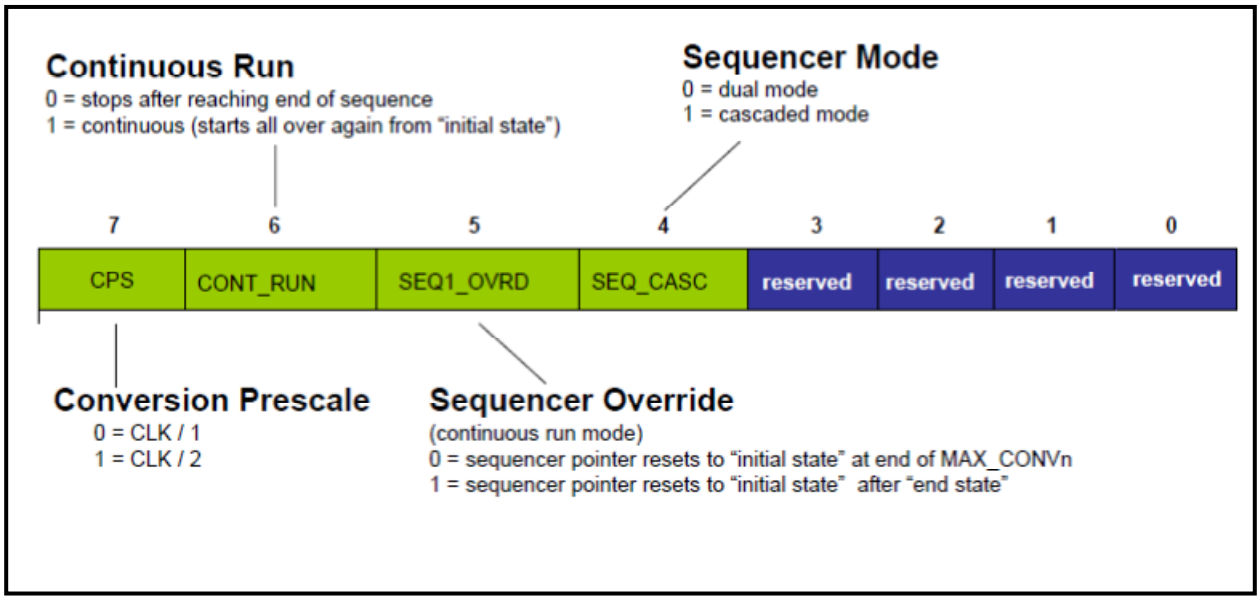


Figura 7.6: *Registro de control ADCTRL1.Bits 0 a 7*

REGISTRO ADCTRL 2.

La mitad superior de registro ADCTRL2 es responsable de controlar el modo de funcionamiento de secuenciador 1.

El **bit 15** "EVB_SOC_SEQ" es un flag que indica si Event Manager B ha provocado la conversión.

Con **Bit14** "RST_SEQ1" podemos resetear la máquina de estado de SEQ1 a su estado inicial. Eso significa que el siguiente disparo comenzará de nuevo desde CHSELSEQ1.

Cuando establecimos nuestra Bit 13 "SOC_SEQ1" a 1 se inicia la conversión por software.

Bits 11 y 10 definen el modo de interrupción de la SEQ1. Podemos especificar si tenemos una solicitud de interrupción por cada "final de la secuencia" (EOS) o cualquier otro (EOS).

Bit 8 "EVA_SOC_SEQ1" es el bit de máscara para activar o desactivar la capacidad de Event Manager A de desencadenar una conversión.

Los bytes más bajos de ADCTRL2 tienen un funcionamiento similar a su mitad superior: controla el secuenciador SEQ2.

Bit 7 es un flag que indica que el pin externo "ADCSOC" ha causado la conversión.

El resto es idéntico al mitad superior.

Las siguientes figuras nos muestra los bits de control del registro de control ADCTRL2.

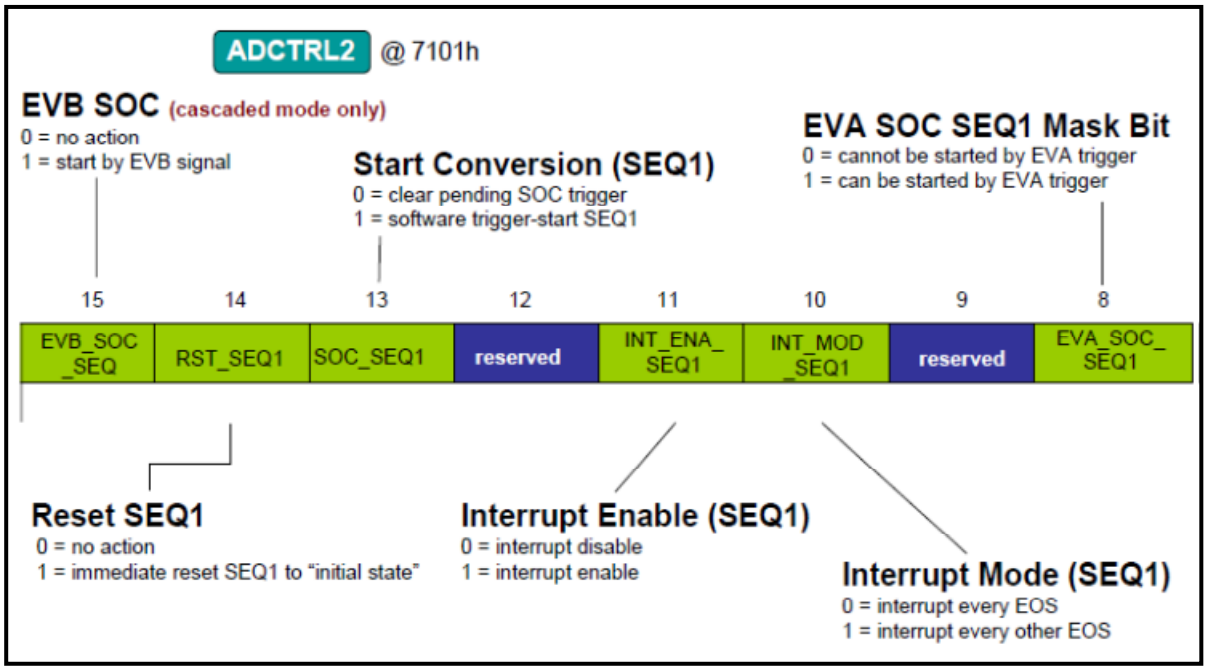


Figura 7.7: *Registro de control ADCTRL2.Bits 8 a 15*

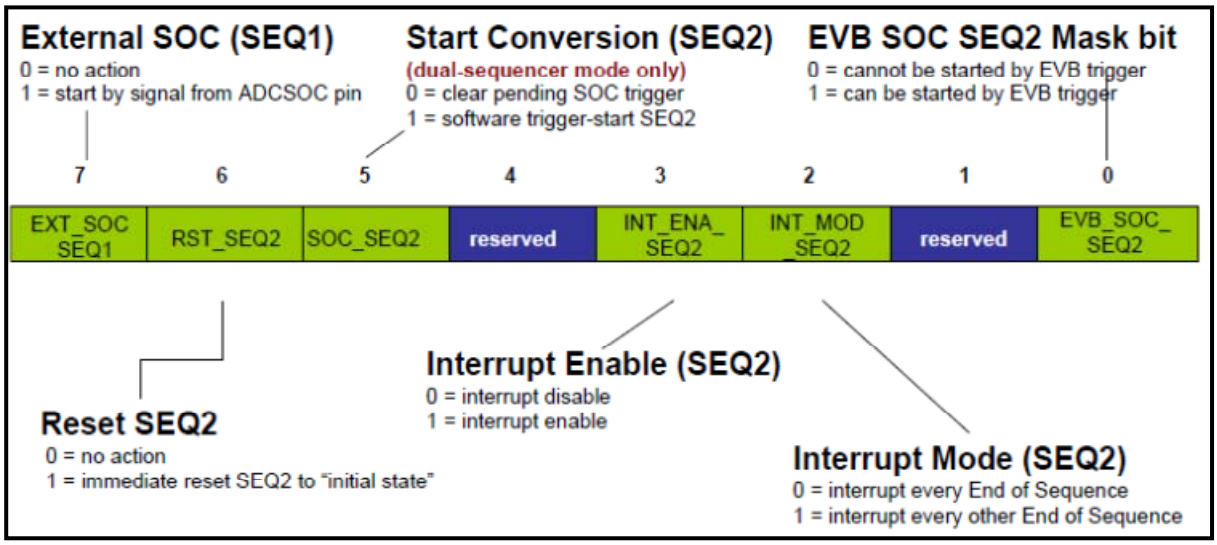


Figura 7.8: *Registro de control ADCTRL2.Bits 0 a 7*

REGISTRO DE CONTROL ADCTRL3

La siguiente figura nos muestra los bits del registro de control ADCTRL3.

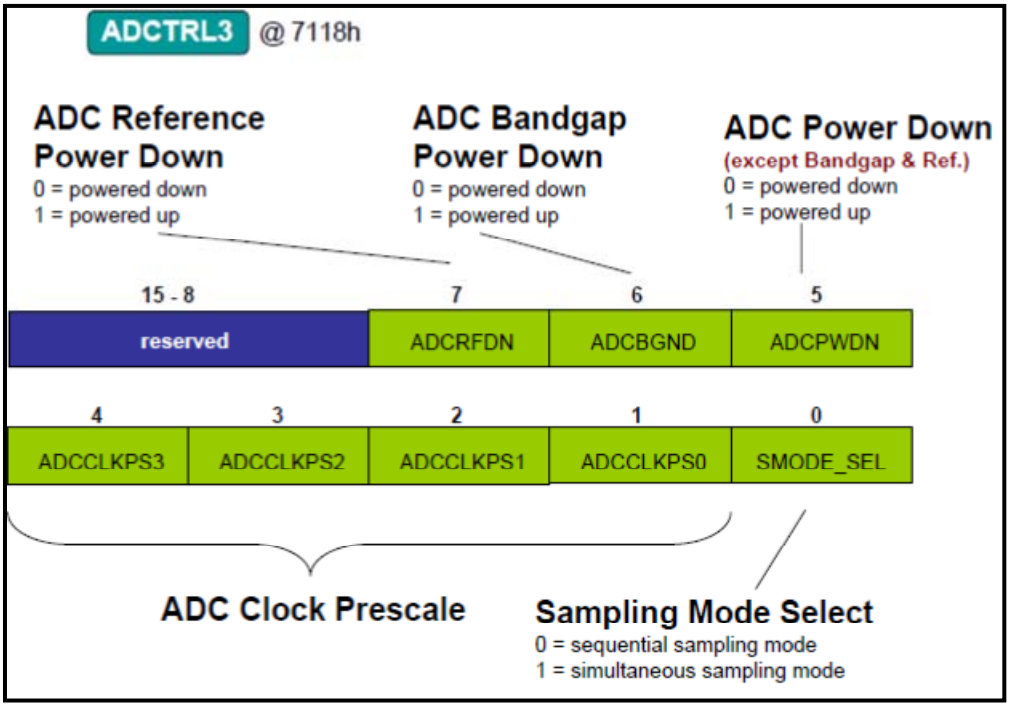


Figura 7.9: *Registro de control "ADCTRL3"*

REGISTRO ADCMAXCONV

En la siguiente figura se muestra el registro del número máximo de conversiones AD.

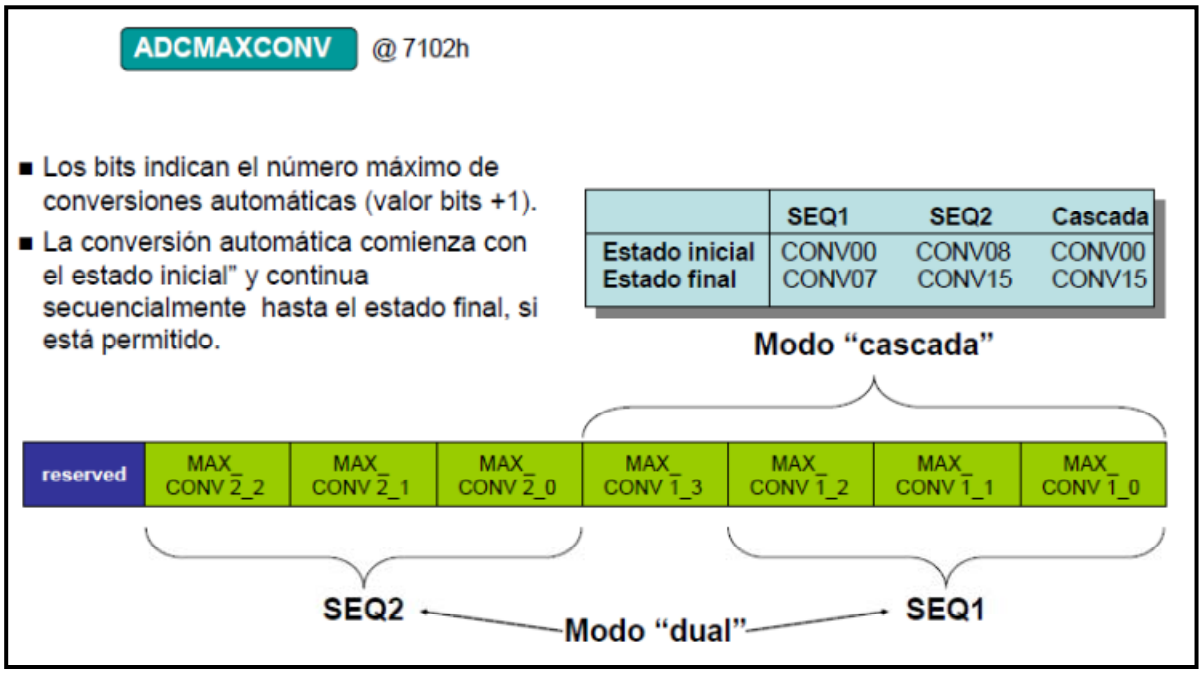


Figura 7.10: *Registro de máximo número de conversiones*

REGISTROS DE CONTROL DE LA SECUENCIA DE CONVERSION

A continuación se muestran los registros para el control de la secuencia de conversión.

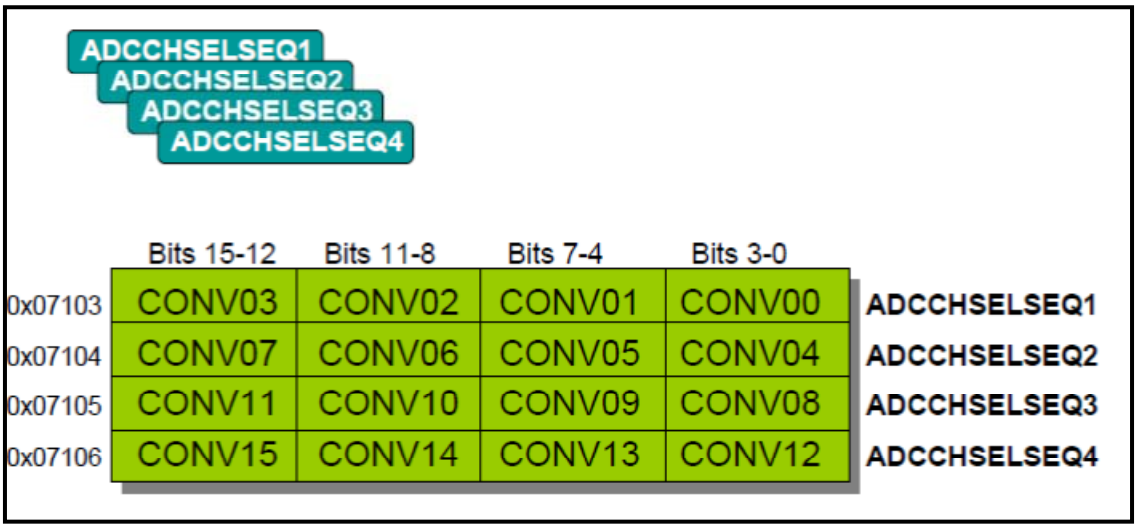


Figura 7.11: *Registro de control de secuencia de la conversión.*

REGISTROS DE RESULTADO DE LA CONVERSION AD

Los siguientes registros son donde almacenamos el resultado de la conversión AD.

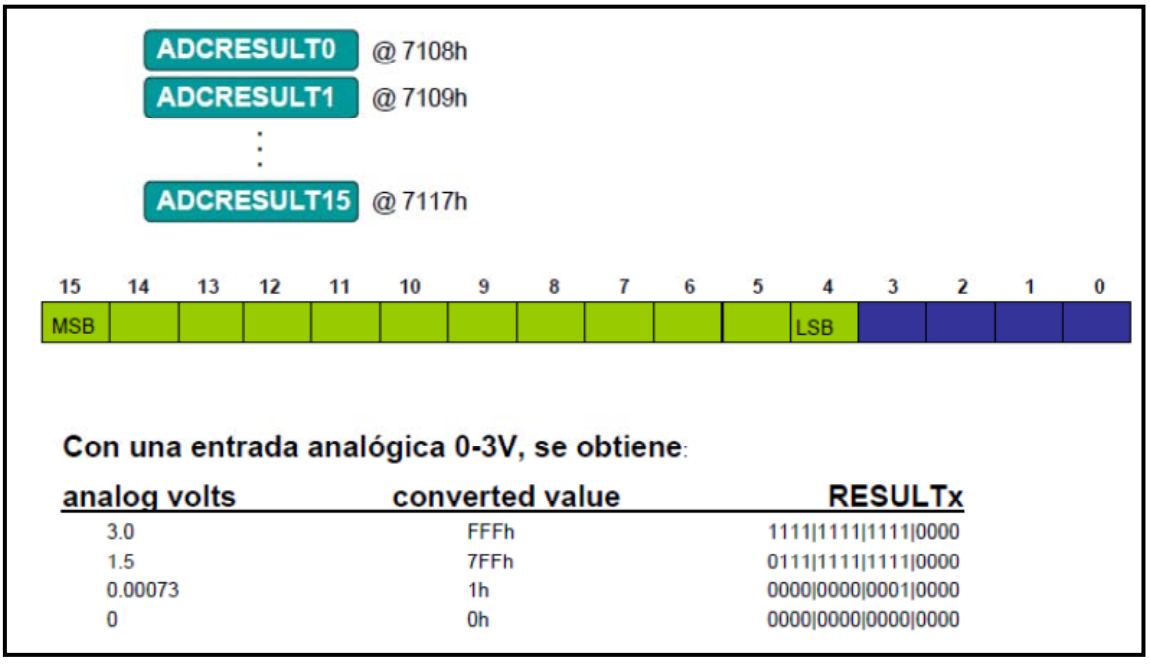


Figura 7.12: *Registro resultado de la conversión ADC.*

¿Como leer el resultado?

Los 12 bits digitales resultados están justificados a la izquierda.

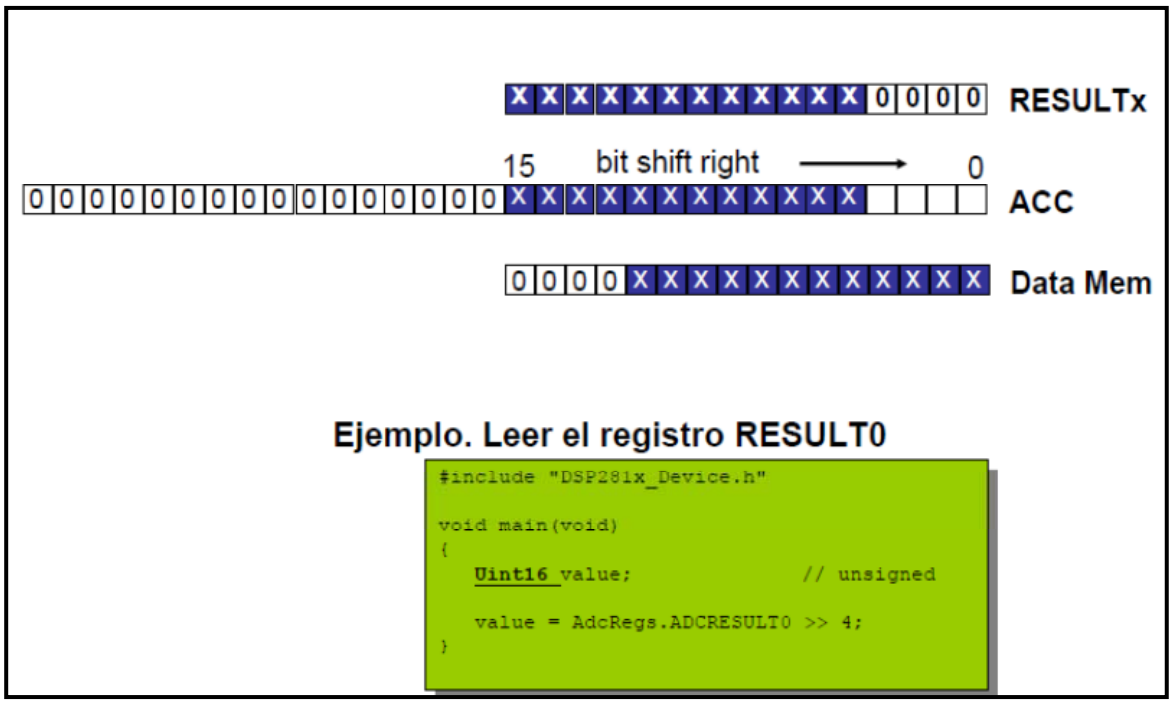


Figura 7.13: *Como leer el resultado*

1.7.7.-Ejemplo del uso del secuenciador.

La figura siguiente nos sirve para entender un sencillo ejemplo del uso del secuenciador descrito a continuación.

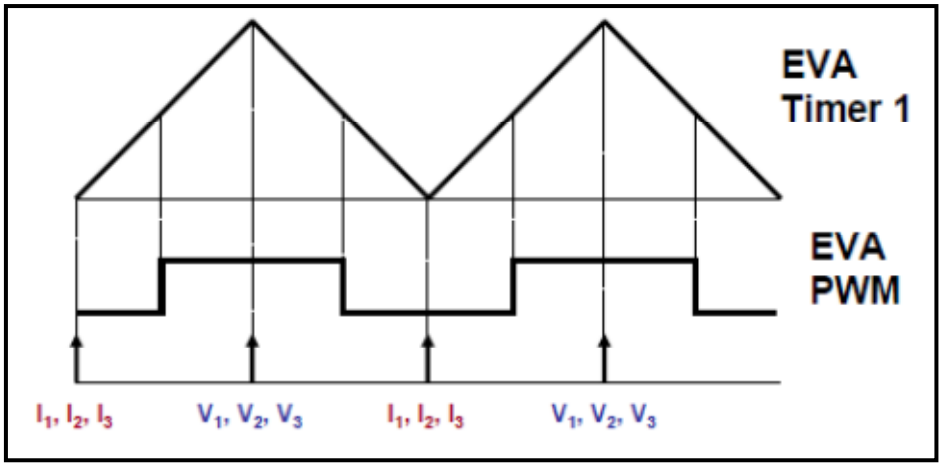


Figura 7.13: *Relación del timer1 con la señal PWM generada.*

•Tres conversiones automáticas (I1, I2, I3) con trigger 1 (Timer underflow)

•Three conversiones automáticas (V1, V2, V3) con trigger 2 (Timer period)

Event Manager A (EVA) y SEQ1 en modo de muestreo secuencial

• MAX_CONV1 =2. Indica que se hacen 3 conversiones aútomáticas en cada etapa.

• Los registros de control de la secuencia de conversión quedan así:

Bits →	15-12	11-8	7-4	3-0	
0x007103	V_1	I_3	I_2	I_1	ADCCHSELSEQ1
0x007104	x	x	V_3	V_2	ADCCHSELSEQ2

• Tras el RESET y la inicialización, el secuenciador SEQ1 espera un trigger.

• Tras este disparo se hacen tres conversiones: CONV00 (I1), CONV01 (I2), CONV02 (I3)

• Se inicializa MAX_CONV1 =2 (a menos que se cambie por software)

• El secuenciador SEQ2 espera un segundo trigger

• Tras este disparo se hacen tres conversiones: CONV03 (V1), CONV04 (V2), CONV05 (V3)

• Al final de las dos etapas los rgistros de resultados tienen los siguientes datos:

RESULT0	I_1	RESULT3	V_1
RESULT1	I_2	RESULT4	V_2
RESULT2	I_3	RESULT5	V_3

•El usuario puede hacer un RESET de SEQ1 por software a la etapa CONV00 y repetirlo todo de nuevo.

• El secuenciador SEQ1 se mantiene en el estado actual mientras espera otro trigger

1.8.- Herramientas de desarrollo de programa

1.8.1 – Introducción

El objetivo de este capítulo es entender las funciones básicas del Code Composer Studio en el ambiente integrado para la familia C2000 del diseño de los DSP´s de Texas Instruments. Esto implica conocer la estructura básica de un programa (denominado proyecto) en C y del ensamblador de archivos de fuente cifrados, junto con las operaciones de compilar, enlazar y volcar al DSP.

El Code Composer Studio es el entorno para el desarrollo del proyecto y para todas las herramientas necesarias para construir un programa para la familia de C2000 (Figura 8.1).

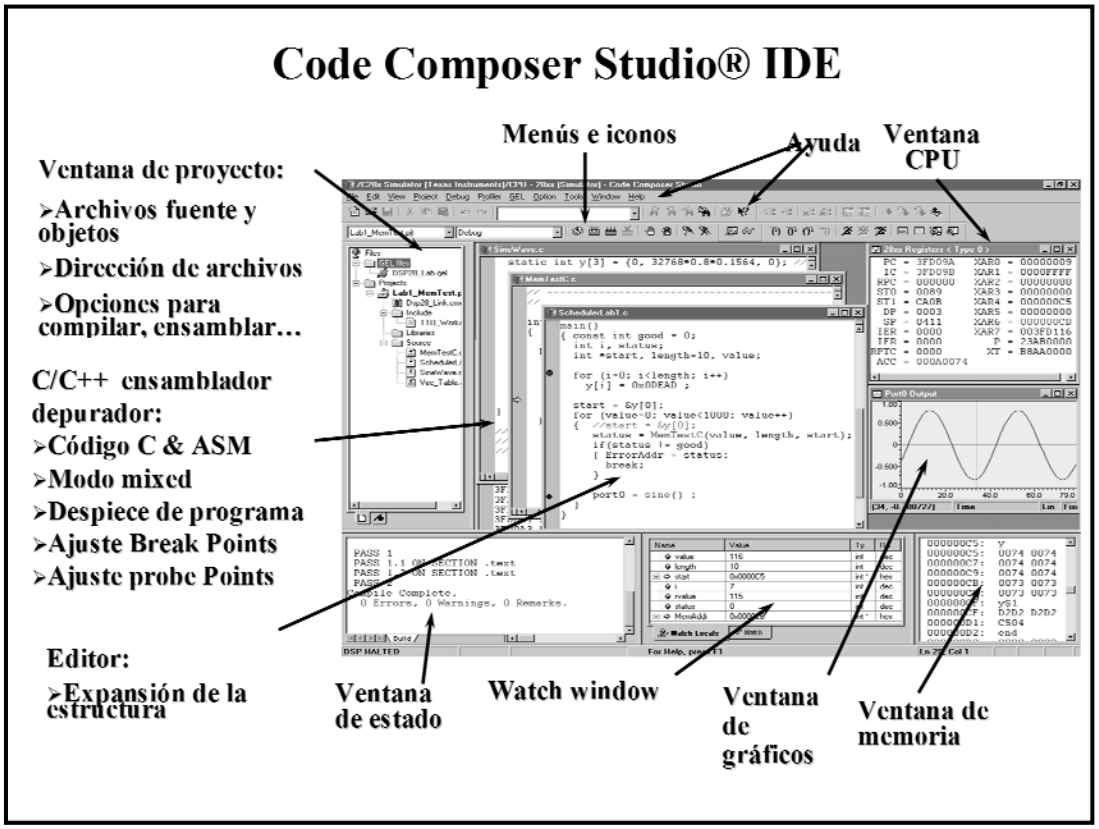


Figura 8.1: Ventana principal del Code Composer Studio.

1.8.2.- El Software

La figura siguiente ilustra el flujo del diseño del software dentro del CCS. Los pasos básicos son: editar, compilar y conectar. El proceso es muy parecido al de otros compiladores integrados en un PC como Microsoft Visual Studio. En este caso, la diferencia principal corresponde a las conexiones con el hardware.

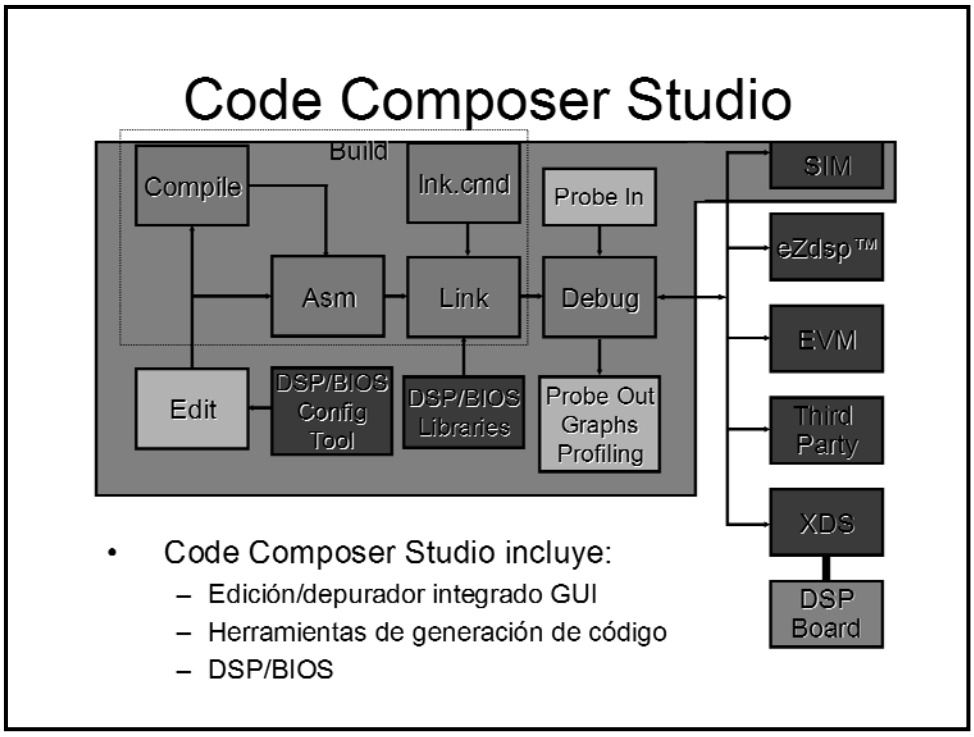


Figura 8.2: Flujo de diseño del software.

Se puede emplear el CCS como un simulador (que funciona en el PC) o puede conectar un sistema real DSP y probar el software en una aplicación real.

Las diapositivas siguientes muestran algunas características básicas del Code Composer Studio y del setup del hardware para los ejercicios del laboratorio que siguen.

1.8.3.- Code Composer Studio

En las siguiente diapositivas se van a ver las características del Code Composer Studio (Figura 8.3), el contenido de los ficheros del programa (Figura 8.4) y algunos de los menús de las opciones del constructor (Figuras 8.5 ,8.6 y 8.7).

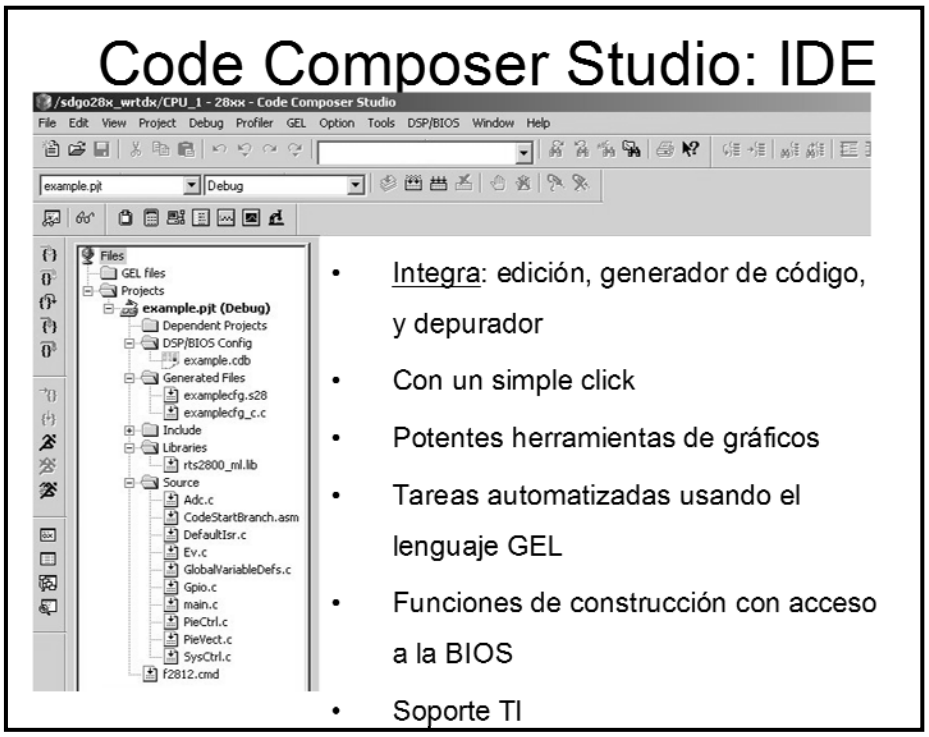
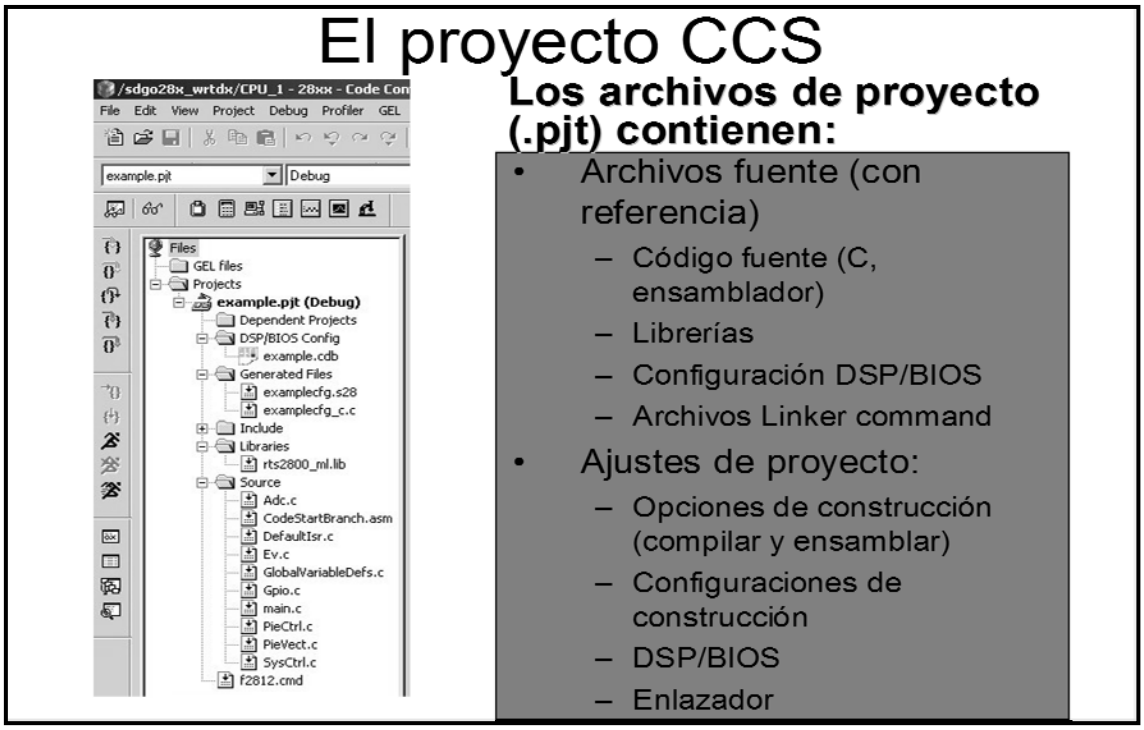


Figura 8.3: Características del CCS



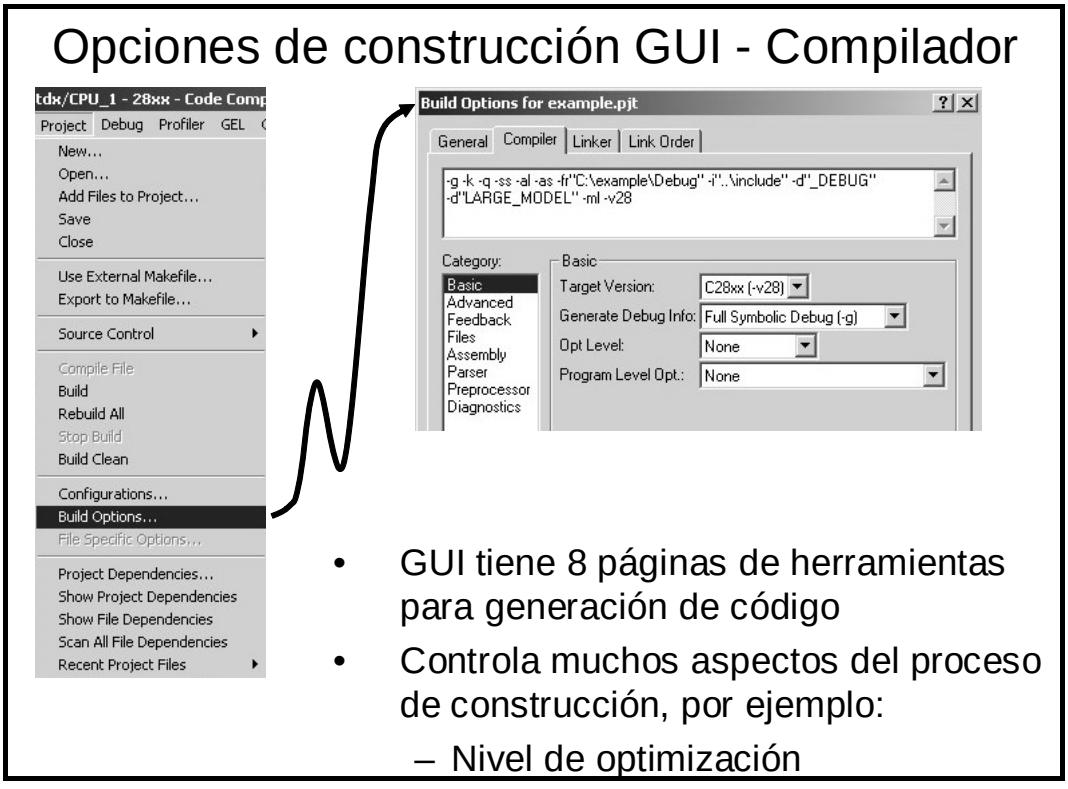


Figura 8.5: Opciones de construcción. Pestaña de compilador

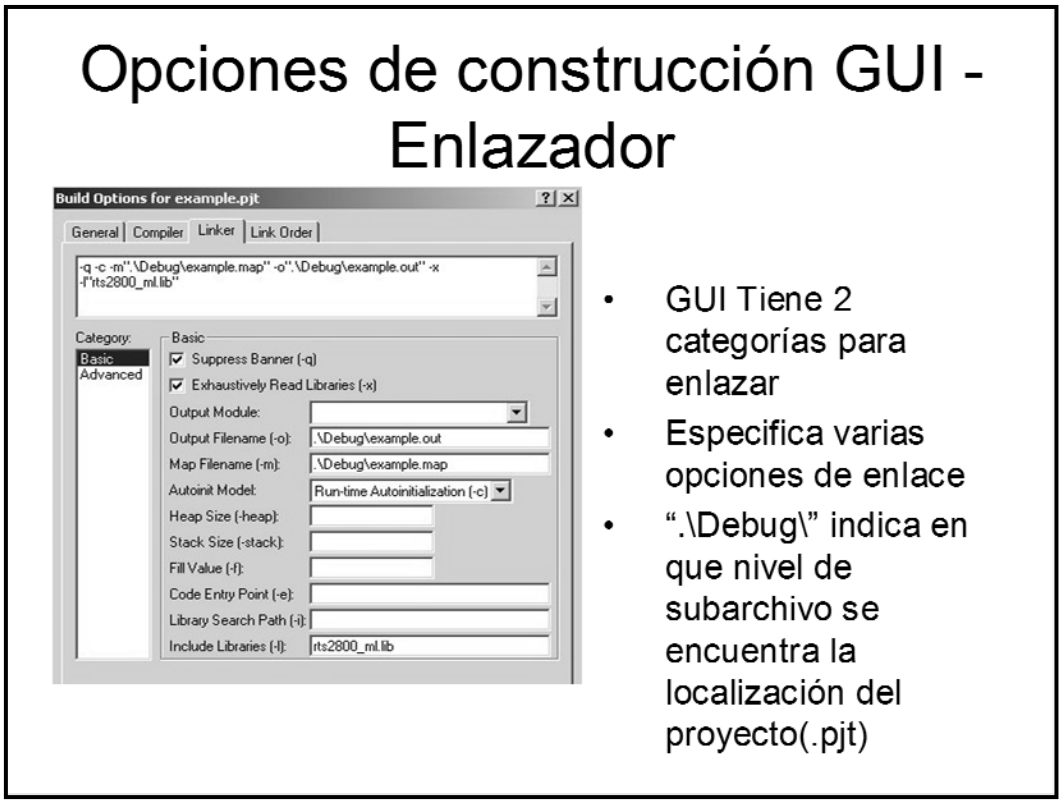


Figura 8.6: Opciones de construcción. Pestaña del enlazador.

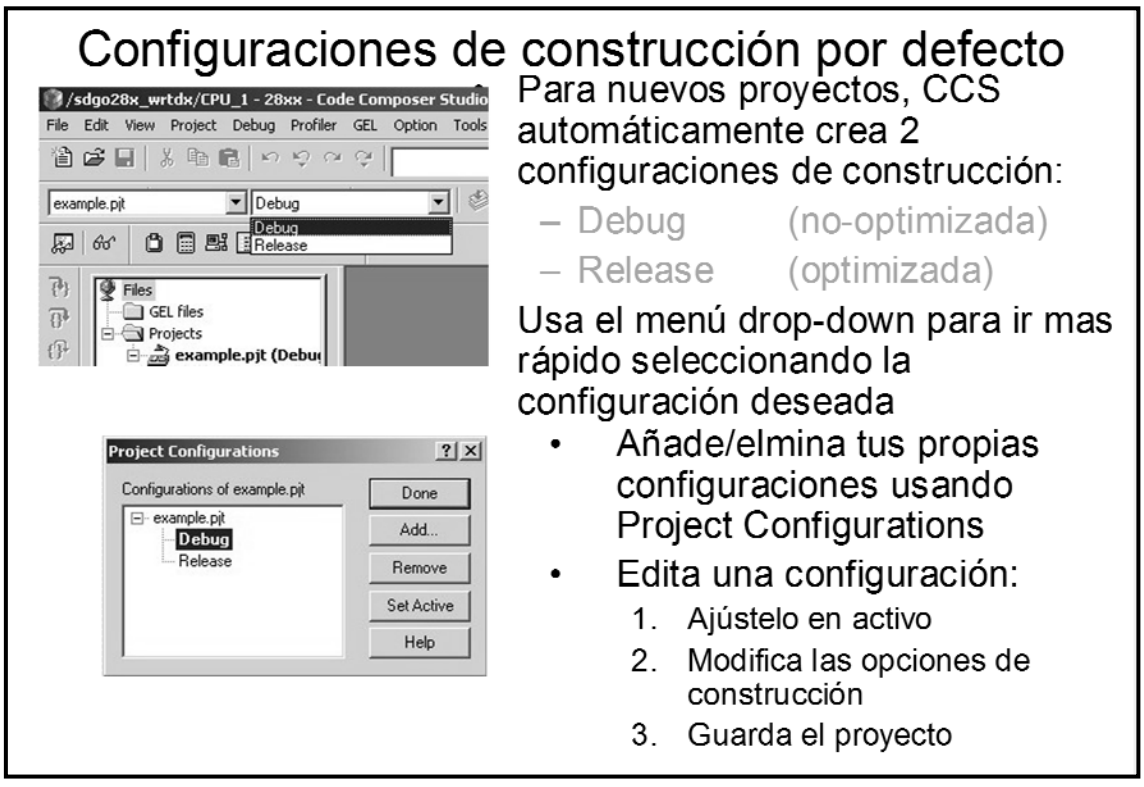


Figura 8.7: Configuraciones para nuevos proyectos.

1.8.4.- Code Composer Studio – Paso a paso

Se va a describir ahora un tutorial paso a paso sobre la creación, compilación y depuración de un programa en el entorno del CCS. Esta parte es una traducción de la ayuda que incluye dicho programa.

Al iniciar el CCS a través del icono de escritorio se observa una pantalla similar a la indicada en la figura 8.8.

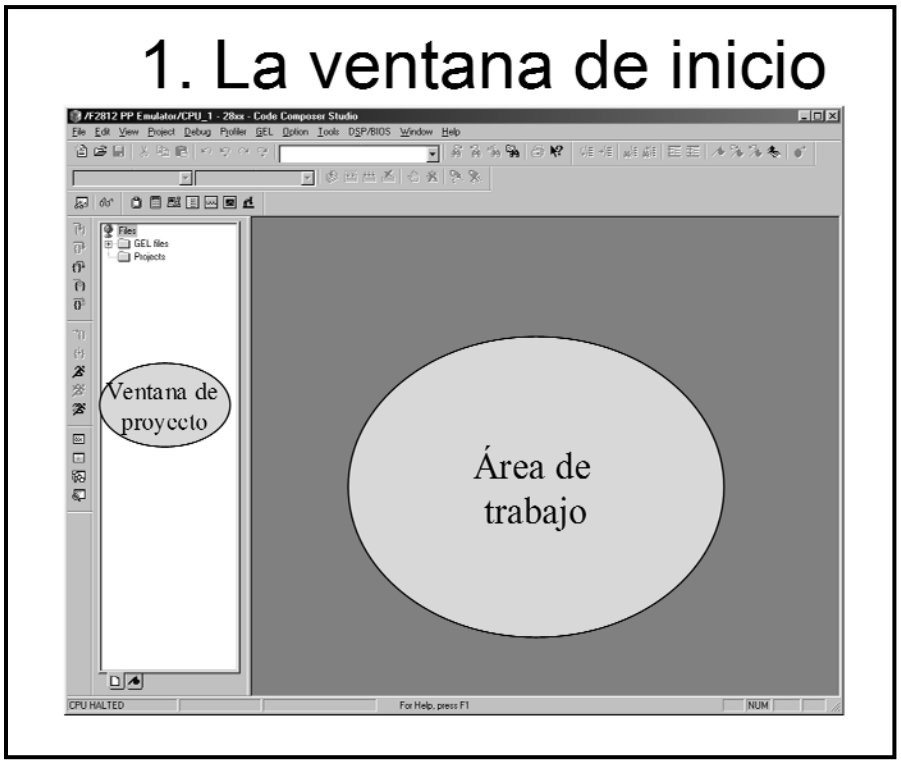


Figura 8.8: Ventana de inicio del CCS

A partir de aquí se van a realizar los siguientes pasos:

- Abrir el Code Composer Studio
- Crear el proyecto de F28x basado en C
- Compilar, conectar, descargar y eliminar errores del programa
- Mirar las variables
- Comparar el funcionamiento en tiempo real y el simple
- Utilizar Breakpoints
- Mirar las partes esenciales del DSP durante la eliminación de errores

1.8.5.- Crear un proyecto

La primera tarea es crear un proyecto. Esto es similar a la mayoría de entornos actuales de diseño, pero con una excepción: hay que definir el tipo de DSP para el que se va a crear el proyecto, que en nuestro caso "TMS320C28xx" (Figura 8.9). El proyecto se genera a si mismo un subdirectorio con el mismo nombre que el del proyecto.

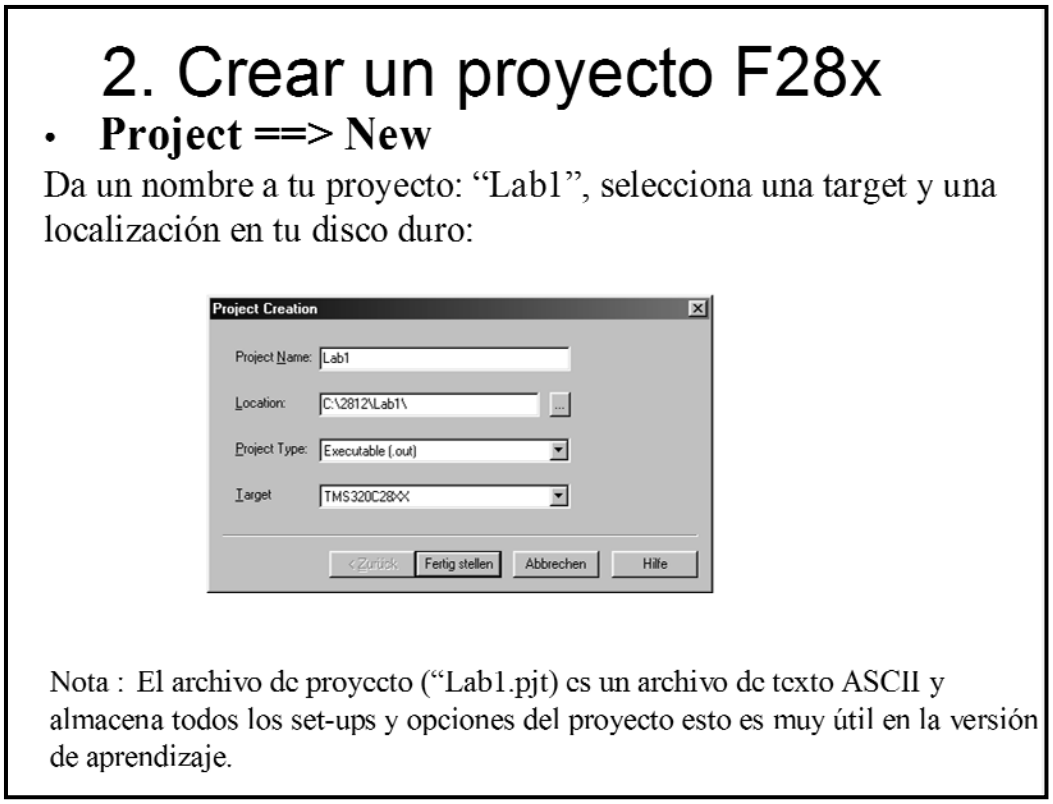


Figura 8.9: Creación de un nuevo proyecto.

Después, escriba el código fuente para su primer uso. El programa de la figura siguiente es una de las tareas más simples para un procesador. Consiste en un lazo sin fin, que cuenta la variable i de 0 a 99, calcula el producto actual de i * i y lo almacena temporalmente en k. Una vez creado el proyecto se deberá guardar con un nombre.

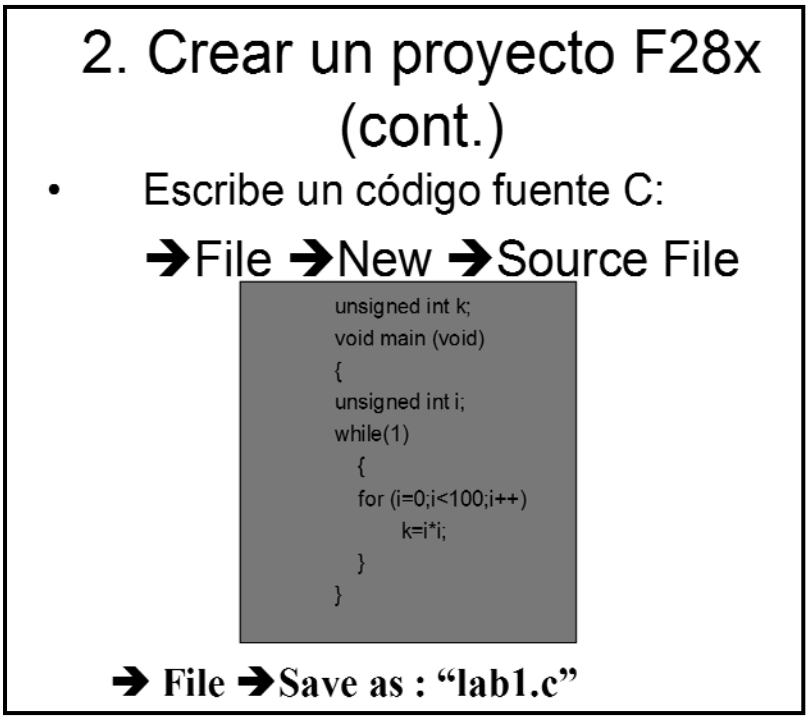


Figura 8.10: Nuevo código fuente, creación y guardado.

Después de escribir el código deberá aparecer algo parecido a la figura 8.11.

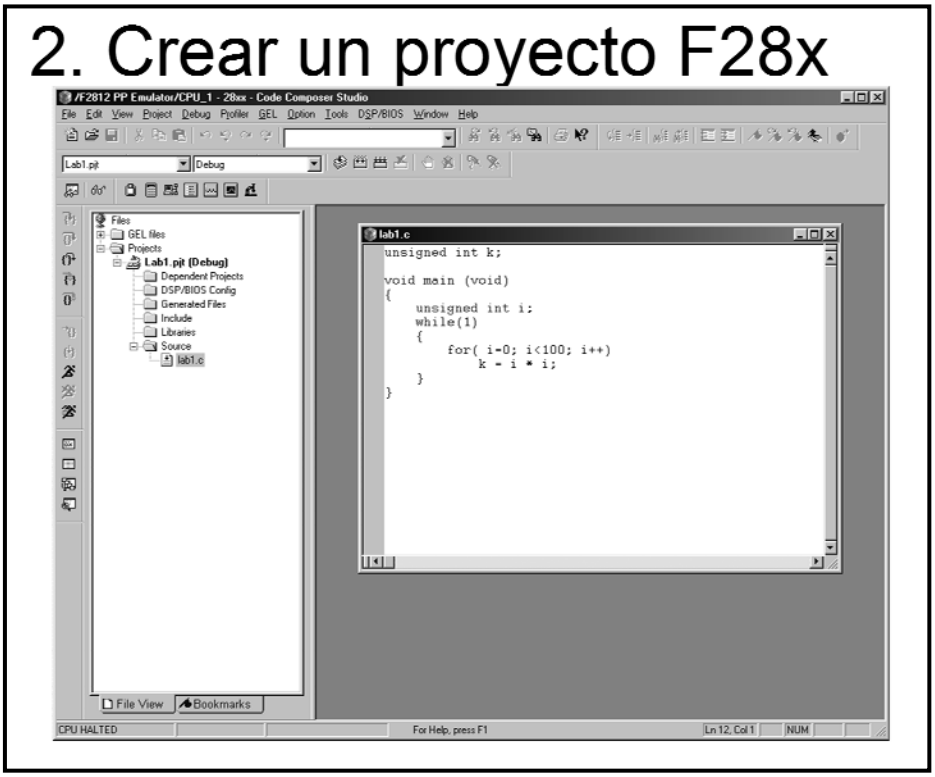


Figura 8.11: Ventana de nuevo proyecto.

El archivo del código debe ser guardado en el disco, pero todavía no forma parte del proyecto. No se realiza esta tarea de forma automática para evitar que todos los archivos que modifiquemos durante la fase del diseño del proyecto fueran agregados al proyecto.

El modo de agregar un archivo a su proyecto debe ser realizado de forma explicita por el programador. Esto no es solamente válido para los archivos del código de fuente, sino también para el resto de los archivos que necesitemos para generar el código automático del DSP.

1.8.6.- Opciones de compilación

Los pasos a seguir para la creación de un nuevo proyecto se resumen en la siguiente figura:

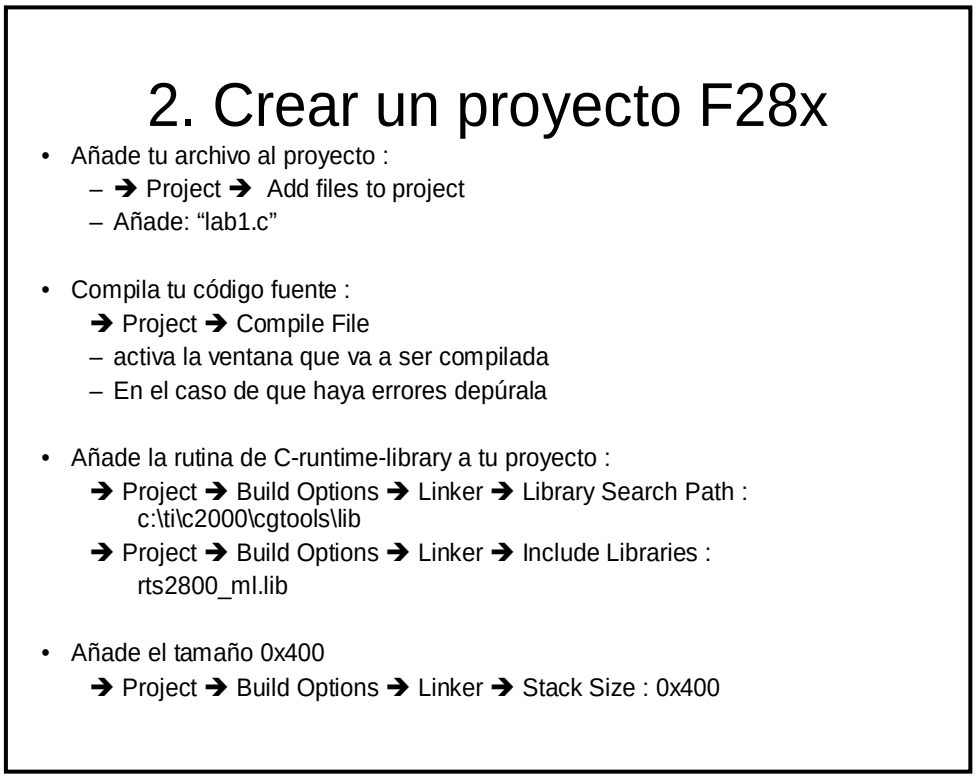


Figura 8.12: Resumen de los pasos para crear un nuevo proyecto.

Por defecto el valor de la pila del compilador viene con un valor no valido, para poner en la pila el valor 0x400 deberemos entrar en las opciones del constructor, pestaña linker y modificar el valor de *Snack size.* Este menú aparece en la figura 8.13.

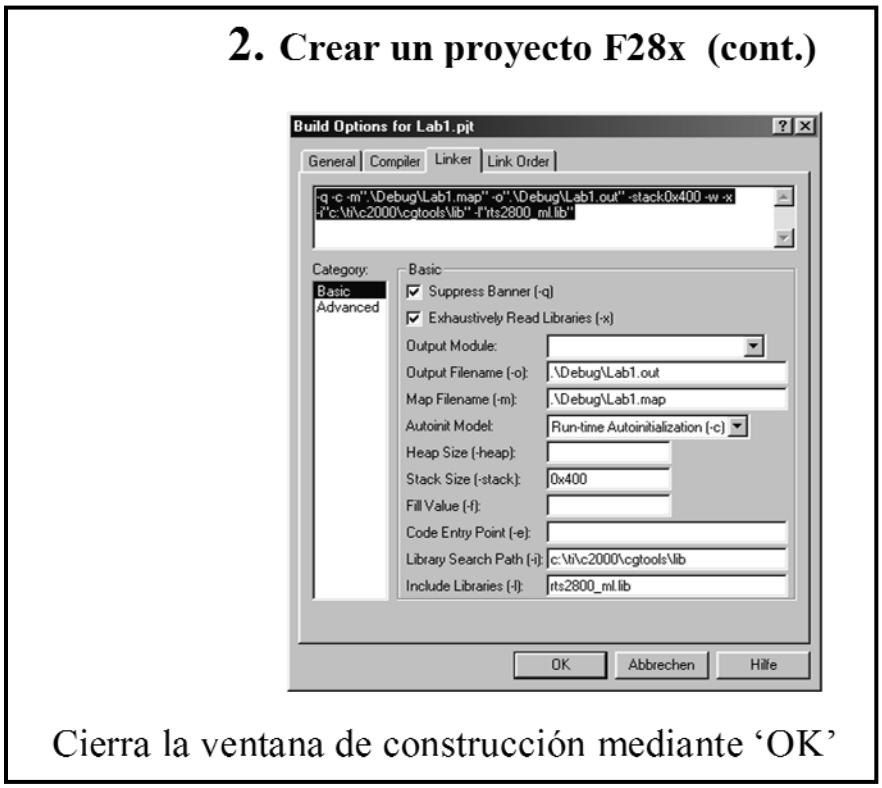


Figura 8.13: Modificación del tamaño de la pila.

1.8.7.- Archivo de comando para el compilador

Ahora hay que indicar al "linker" como unir el programa, con la librería y en que posición de memoria del DSP hay que volcar el programa. Para ello se emplea un "archivo de comandos, con extensión .cmd. Este archivo indica como se conectan las partes físicas de la memoria del DSP con las secciones lógicas creadas por el software. La figura 8.14 lo aclara.

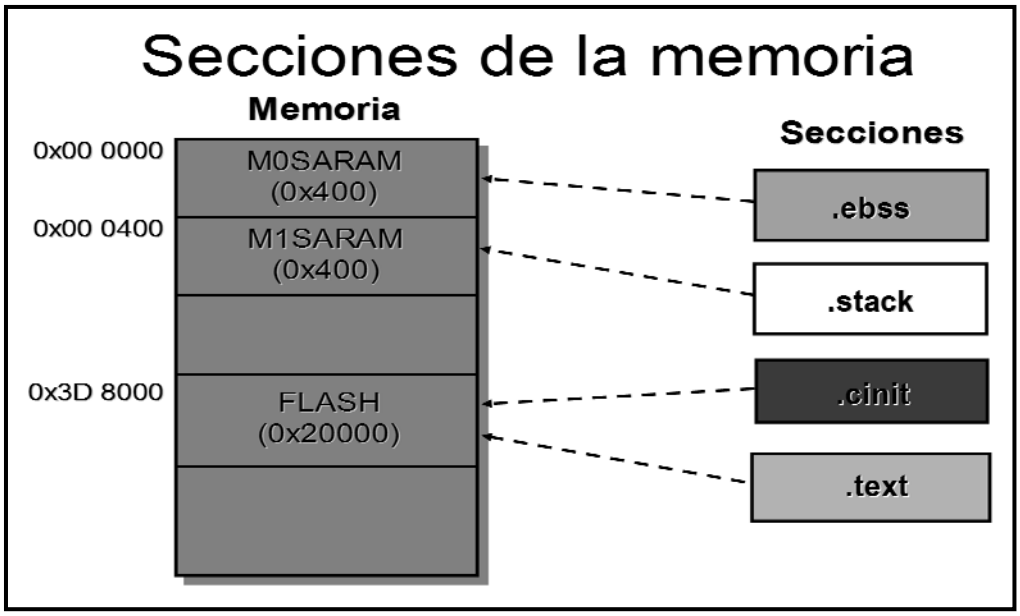


Figura 8.14: Secciones de memoria.

El archivo que se va a usar es el fichero F2812_EzDSP_RAM_lnk.cmd, que ha sido diseñado por Texas Instruments y es parte del paquete de software del Code Composer Studio. Una parte de este fichero está en la figura 8.15.

Archivo Linker Command

```
MEMORY
{
  PAGE 0:        /* Program Space */
    FLASH:    org = 0x3D8000,  len = 0x20000

  PAGE 1:        /* Data Space */
    M0SARAM:  org = 0x000000,  len = 0x400
    M1SARAM:  org = 0x000400,  len = 0x400
}

SECTIONS
{
    .text:    >      FLASH         PAGE 0
    .ebss:    >      M0SARAM       PAGE 1
    .cinit:   >      FLASH         PAGE 0
    .stack:   >      M1SARAM       PAGE 1
}
```

Figura 8.15: Parte del fichero del enlazador.

Al final de todo el procedimiento de enlazar y conectar uno o más ficheros objeto (* obj) se genera un archivo de salida (* out), que no sólo contiene el código ejecutable para el DSP sino que además contiene la información usada para eliminar errores, para uso de la memoria Flash del DSP y para otras tareas básicas. También se genera un fichero de recursos con extensión *.map que incluye las posiciones de memoria ocupadas, los símbolos empleados por el compilador y otros elementos que informan del uso de recursos empleados (Figura 9.16).

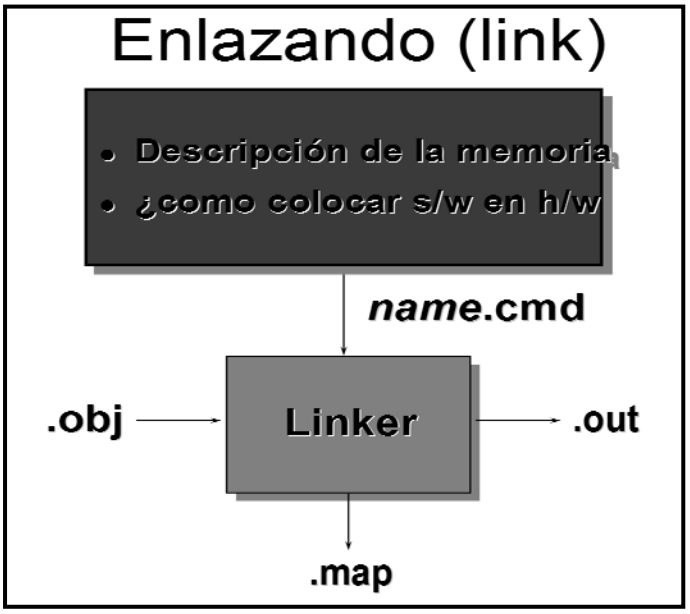


Figura 8.16: Enlazdor y ficheros de salida

1.8.8.- Descargar el código en el DSP

El siguiente paso es volcar el código en el F2812. Se puede hacer de forma automática tras la compilación o hacerlo de forma manual. Para hacerlo de forma manual se seguirán los pasos de la figura 8.17, en la nota se explica la forma de hacerlo de forma automática.

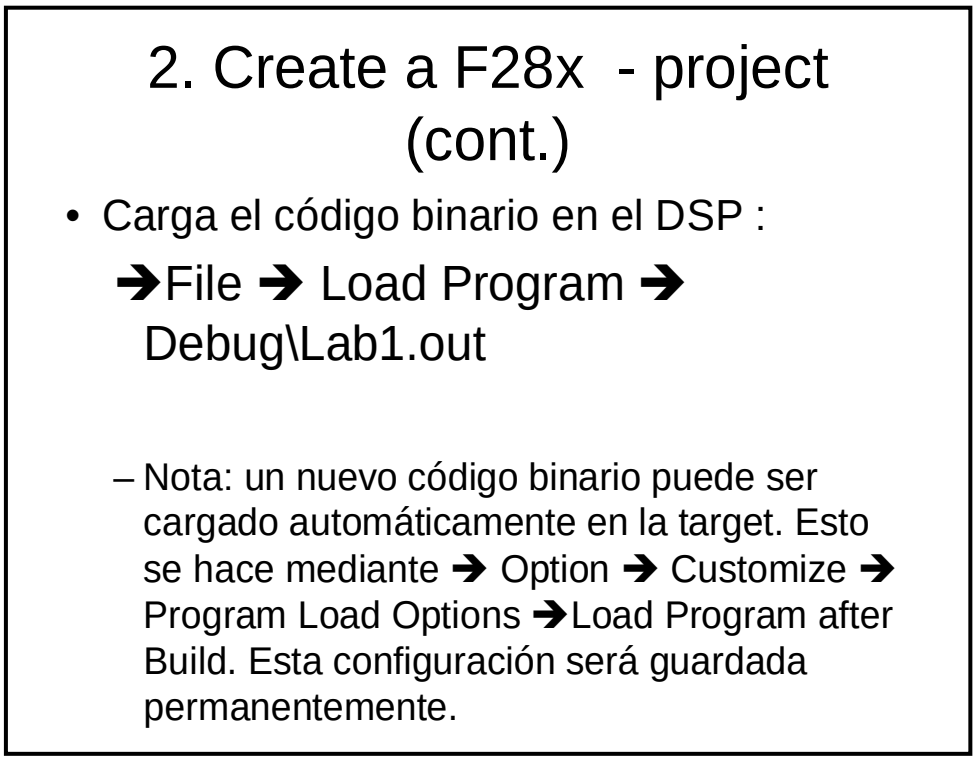


Figura 8.17: Cargar programa en el Dsp

Cuando se ejecuta este primer programa, no hay actividad del hardware que se verá. ¿Por qué no? Bien, nuestro primer programa no utiliza ninguna de las unidades periféricas del DSP.

Si se ejecuta en la ventana de comandos la opción Debug→Go Main (Figura 9.18), el DSP actualiza el código hasta la función main() y una flecha amarilla muestra la posición actual del contador de programa (PC).

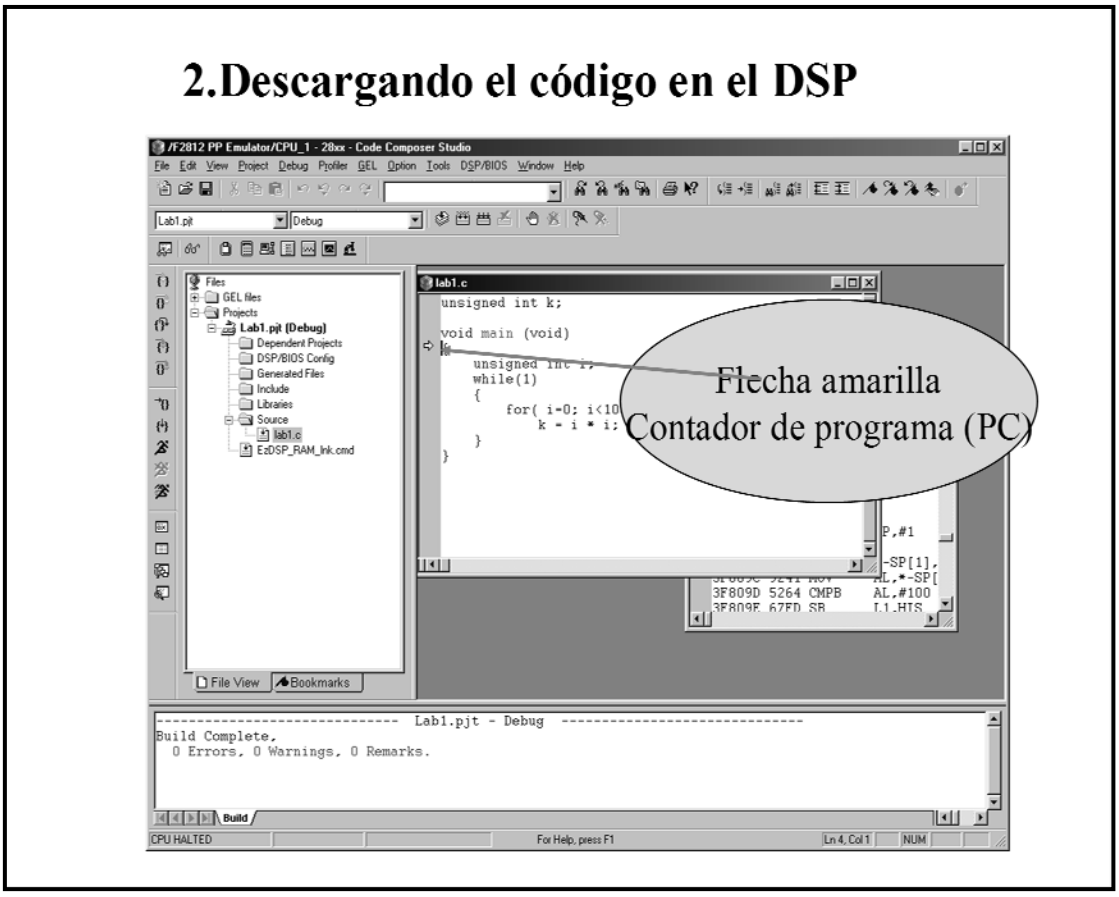


Figura 8.18: Funcion Go Main, el programa se dirige a la función main

Ahora queda ejecutar el código con la tecla F5 o seleccionar la opción Run en el menú Debug. A partir de aquí se puede realizar el proceso de depuración o ejecución paso a paso, mirar variables de memoria, añadir breakpoints, etc…

Para mirar las variables del programa, podemos utilizar una ventana dedicada llamada "watch Window". Ésta es probablemente la ventana más usada durante la fase de prueba de un proyecto del software. Para que esta ventana aparezca se seleccionara la opcion wacht window del menú view (Figura 8.19). El aspecto de esta ventana es el de la figura 8.20

4. Mira tus variables

- Abre la Watch Window :
  ➔ View ➔ Watch Window

- La variable 'i' es ahora visible dentro de la ventana "Watch Locals".
- Para ver la variable global 'k' necesiteremos añadirla manualmente. Esto puede hacerse en l ventana 'Watch 1'. En la columna 'name' introduciremos 'k' y en la segunda linea 'i'.
  – Nota: otra manera es marcar las variables dentro del código fuente con el botón del ratón derecho y seleccionar "Add to watch window"

- Nota: con la columna 'radix' podemos cambiar el formato de los datos seleccionando entre decimal, hexadecimal, binario, etc.

Figura 8.19: funcion match window, nos permite ver el valor de las variables.

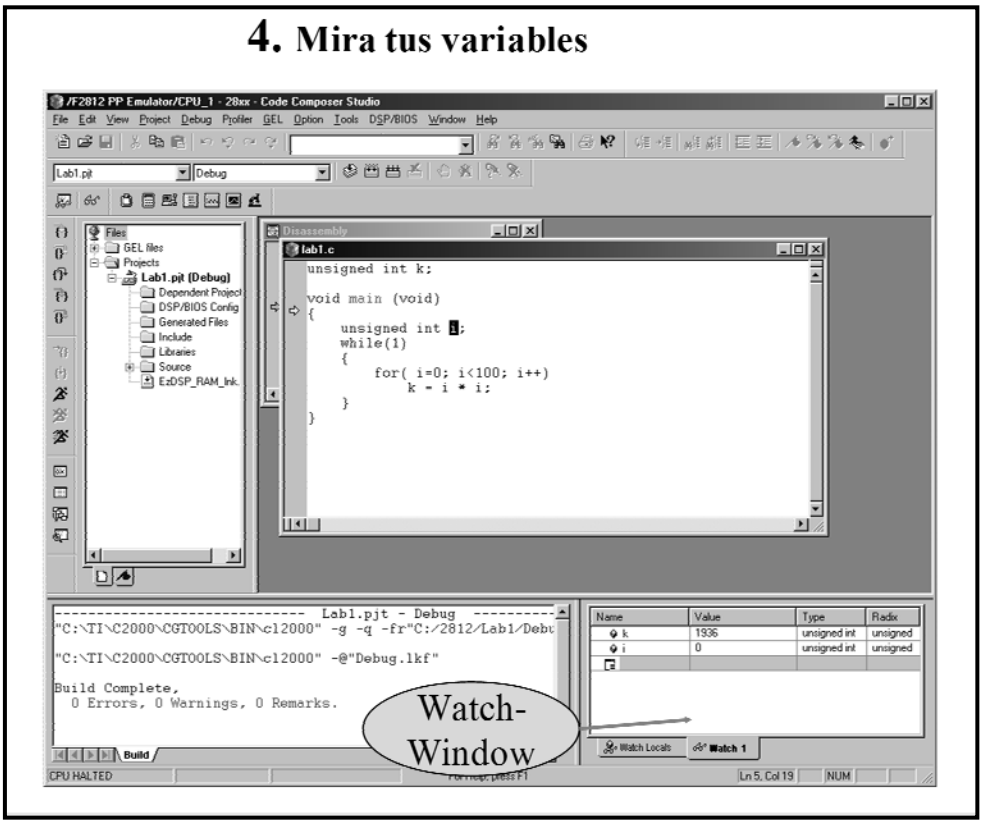


Figura 8.20: Ventana del visualizador de variables.

Otra parte útil de una sesión de debug es la capacidad de eliminar errores del código en porciones y de funcionar el programa para algunas instrucciones, para ello existen una serie de opciones que se indica en la siguiente figura.

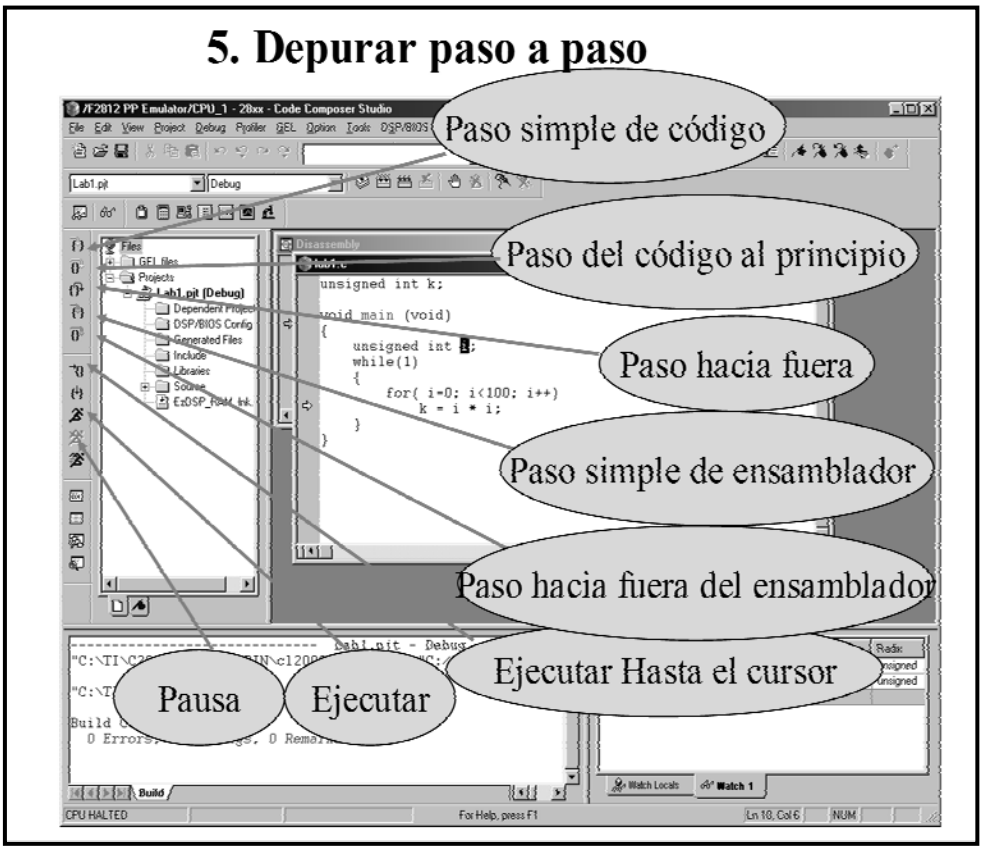


Figura 8.21: Esquema de las funciones de depuración del código.

Cuando se quiera hacer funcionar únicamente unas cuantas líneas de código o que se pare la ejecución en un determinado punto, es necesario recurrir a los "breakpoint" . Para añadir un breakpoint hay que seguir los pasos indicados en la figura 8.22. Los breakpoints aparecerán mediante un punto rojo en el código del programa (Figura 8.23).

6. Añadir un Breakpoint

- Ajustar el Breakpoint :
  - Coloca el cursor en Lab1.c en la línea : k = i * i;
  - Haz click con el botón derecho del ratón y selecciona 'Toggle Breakpoint'
  - La línea será marcada con un punto rojo (= breakpoint activo)

  Nota : muchos comandos de Code Composer Studio pueden ser activados con el teclado (mira el manual o la ayuda)

- Resetea el programa
  ➔ Debug ➔ Reset CPU
  ➔ Debug ➔ Restart

- Ejecuta en tiempo real
  ➔ Debug ➔ Run (or F5)
- El DSP se parará cuando llegue al breakpoint activo
- Repite 'Run' y mira tus variables
- Elimina tu breakpoint (otra vez Toggle).

Figura 8.22: Añadir un breakpoint.

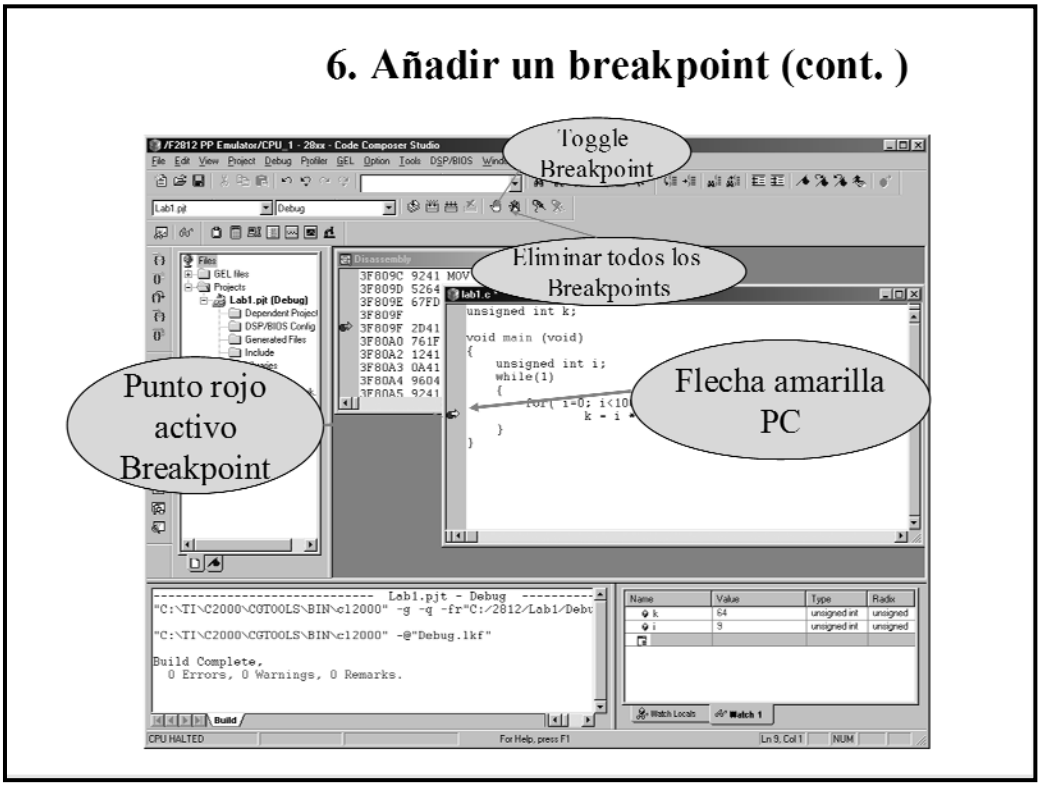


Figura 8.23: Continuación de cómo añadir un breakpoint.

También existen otros tipos de comandos en el entorno CCS para ver los registros de estado, los registros de interrupciones, etc. Esta opción está disponible en el menú View (Figura 8.24 y 8.25).

8. Otros comandos de View

- El menú View incluye las ventanas mas usadas de monitor y control de DSP
- ➔ View ➔ Registers ➔ Core
- ➔ View ➔ Registers ➔ Status

  - Haga click con el botón derecho del ratón dentro de la nueva ventana y seleccione 'Float in Main Window'
  - Doble click en la línea 'ACC' y modifique el valor dentro del Accumulator ACC

Figura 8.24: Mas comandos de la opción de visualización.

8. Otros comandos de View (cont.)

- Para ver tanto el código ensamblador y el código fuente de C:
- Click en el botón derecho del ratón dentro de "Lab1.c" y seleccionar "Mixed Mode"
- La instrucción del código ensamblador es generada por el compilador ir añadida en color gris.

Figura 8.25: Comandos del visualizador.

Existe la opción de observar el código ensamblador y el código C a la vez. Para ello hay que seleccionar la opción Mixed Mode pulsando en el botón derecho sobre el código fuente. El aspecto del CCS será el mostrado en la figura 8.26.

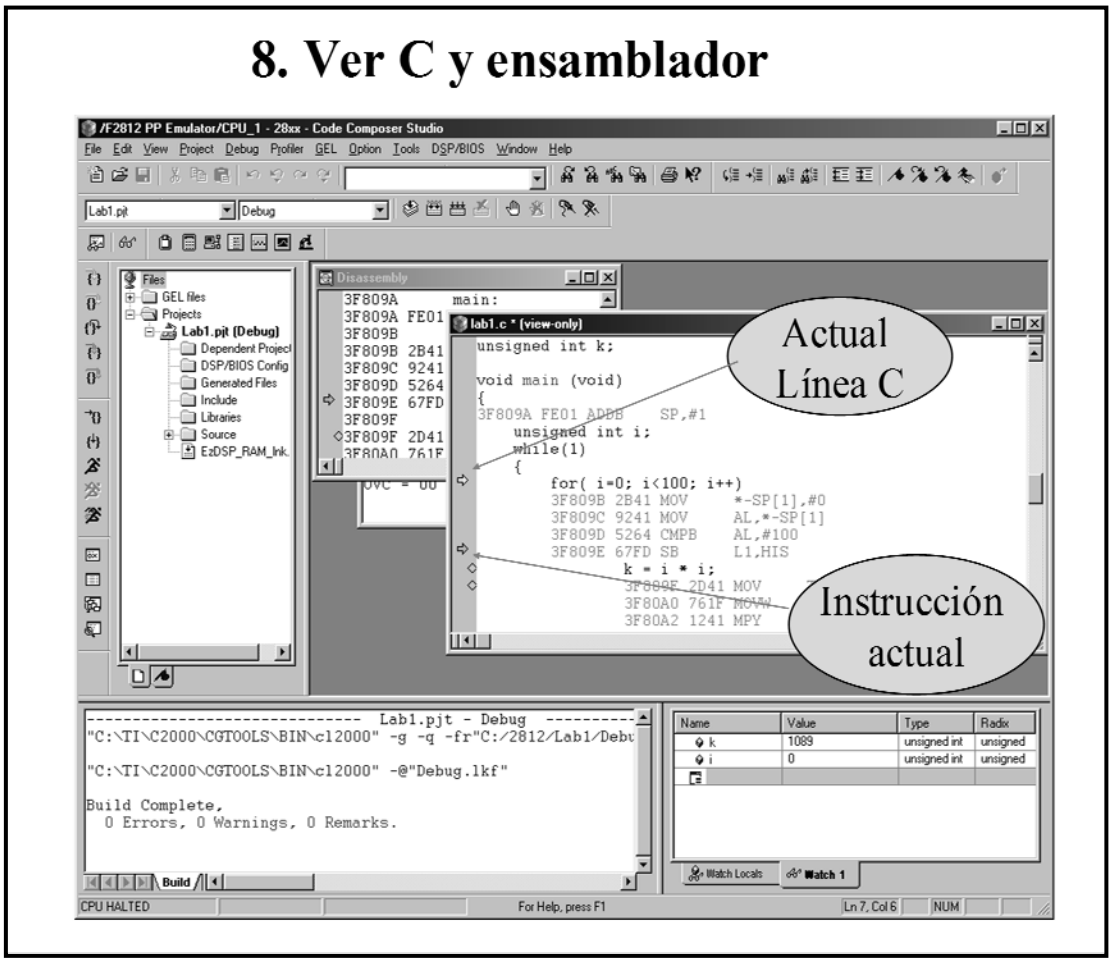


Figura 8.26: Visualización mixta, código C y código maquina.

1.9.- FICHEROS DE AYUDA A LA PROGRAMACIÓN

1.9.1.- Introducción

Hay que hacer especial referencia a estos ficheros que proporciona el fabricante y facilitan la programación a la vez que permiten realizar un proyecto de forma mucho más cómoda.

Estos archivos, que incluyen ficheros cabecera, ficheros de comandos y liberáis de funciones básicas, pueden ser fácilmente incorporados en un nuevo proyecto o a uno ya existente, y el usuario puede seleccionar y elegir las funciones de los ejemplos de código que desee y desechar el resto.

Estos ficheros no proporcionan una forma de escribir C, sino que son una ayuda a la hora de localizar variables o de completar registros de datos, configurar periféricos, etc. Si es importante que el usuario tenga un conocimiento básico de programación en C.

A continuación se describe como se instalan estos archivos para ser utilizados. También se ha incluido la descripción del árbol de ficheros de uno de los ejemplos que se incluyen.

Posteriormente se describe el flujo genérico de un programa y donde se debería incluir el código específico de la aplicación, para acabar con un apartado dedicado a la creación de un proyecto nuevo que incorpore estos ficheros.

1.9.2.- Instalación

Puesto que son proporcionados por el fabricante, este paquete de ficheros incorpora una utilidad de instalación que tras ser ejecutada genera la siguiente estructura en la carpeta C:\tidcs\C28\DSP281x \ <version> que se observa en la figura 9.1.

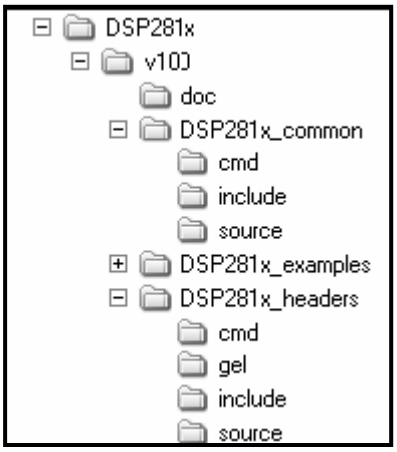


Figura 9.1: Estructura de archivos

Las carpetas y subcarpetas que se generan tras la instalación son las siguientes:

- */doc,* incluye documentos, incluido el historial de revisiones desde la versión anterior.
- */DSP281x_common,* incluye una fuente común a archivos compartidos a través de una serie de proyectos dsp2812, por ejemplo ilustrarte para llevar a cabo las tareas usando el fichero de cabecera dspf2812.

  ./common/cmd, incluye ficheros de asignación de memoria.

  ,/common/include, contiene los ficheros comunes de extensión .h

  /common/source, alberga los ficheros de extensión c

- */DSP281x_examples,* incluye ejemplos de cómo configurar muchos de los periféricos del F2812

- */DSP281x_headers,* incluye los archivos necesarios para incorporar los ficheros cabezera en un proyecto

  /headers/cmd, incluye los archivos de comandos para el linker.

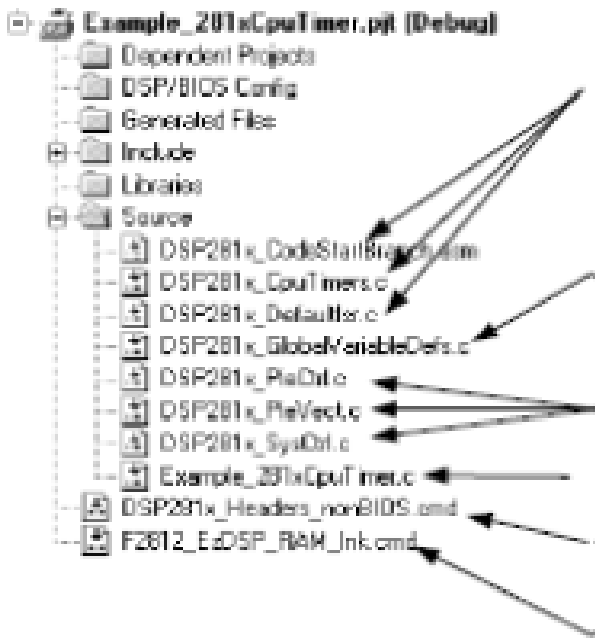
  /headers/source, incluye los ficheros fuente que requiere el fichero cabecera

  /headers/include, contiene los ficheros necesarios para los periférico del f2812.

1.9.3.- Abrir un ejemplo.

Los pasos a seguir para utilizar cualquier ejemplo son los mismos que cuando se ha creado un proyecto. Localizar el fichero .pjt y cargar, compilar y volcar al DSP. A partir de aquí se puede mirar variables, poner breakpoints o examinar la memoria.

Cada uno de los programas de ejemplo tiene una estructura muy similar. Esta estructura incluye su código fuente propio, archivos de cabecera y archivos de comandos para el enlazador (Figura 9.2).



Común (compartido) Código fuente. Utilizado por más de un ejemplo. Contener funciones genéricas para la creación de peripherls a un determinado estado o funciones que pueden ser útil para su reutilización en diferentes aplicaciones.

Este archivo de origen es necesario para utilizar el periférico DSP281x archivos de cabecera.

Shared Source Code

Ejemplo

Requerido para el archivo especifico de cabecera

Bloque de memoria

Figura 9.2: Estructura de archivos de un proyecto

Los archivos cabecera dependen de los que se hayan incluido en el código fuente con la directiva #include. En cualquier ejemplo, los ficheros .h incluidos pueden ser parecidos a la figura 9.3, donde se pueden ver los diferentes nombres de dichos ficheros, que corresponden a los diferentes periféricos del F2812.

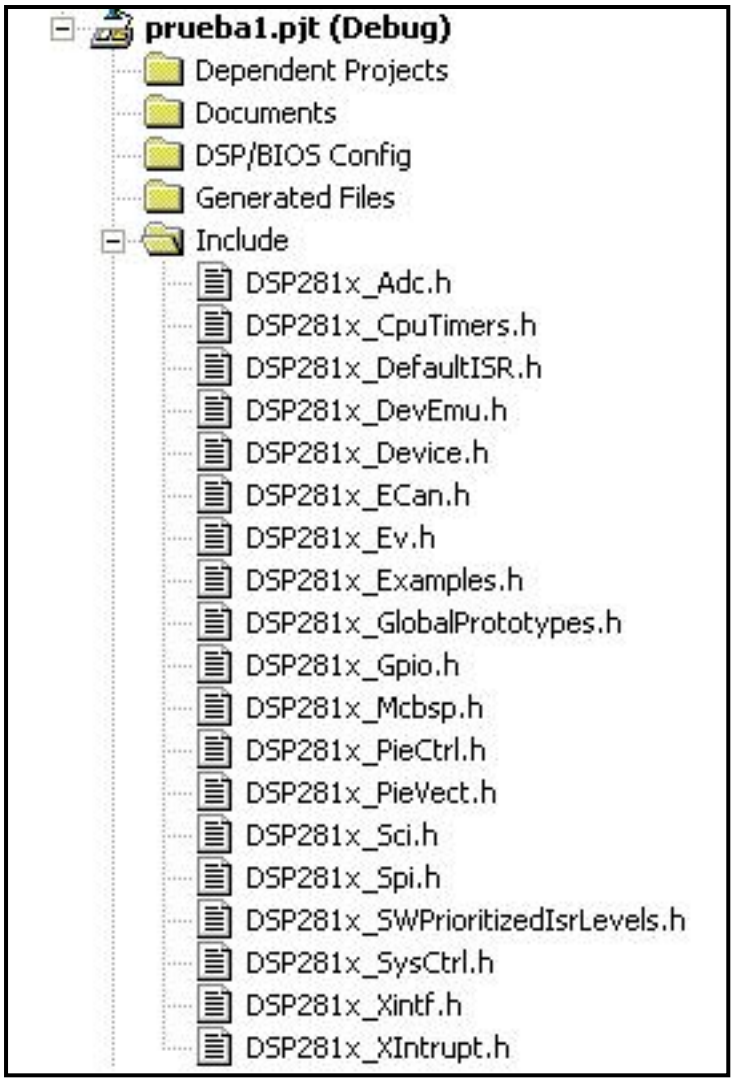


Figura 9.3: Estructura de archivos del proyecto prueba1

Todos estos archivos *.h son automáticamente incluidos cuando se escriben las líneas de código indicadas mas abajo en el fichero .c que incluye la función main().

#include “DSP281x_Device.h”

#include “DSP281x_Examples.h”

Entre los archivos que son necesarios para programar se encuentran los siguientes:

- **Ficheros .h:**

DSP281x_Device.h: incluye todos los periféricos requeridos por los ficheros de cabecera e incluye macros específicos y declaraciones. Esta incluido en la carpeta DSP281x_headers\include.

DSP281x_Examples.h: esta cabecera define los parámetros que son usados por el código ejemplo. Es requerido por algunas cabeceras comunes a algunos archivos. Se encuentra contenido en la carpeta DSP281x_common\include.

- **Ficheros .c:**

DSP281x_GlobalVariableDefs.c: cualquier proyecto que incluya cabeceras de periféricos requiere este archivo, contiene la declaración de la estructura de registros de las variables de los periféricos y la asignación de secciones de datos.

- **Ficheros .cmd:**

F2812_EzDSP_RAM_lnk.cmd: contiene el mapa de memoria que asigna la localización de la SARAM.

DSP281x_Header_nonBIOS.cmd: es un archivo enlazador, se usa para asignar las variables en los archivos de cabecera en un proyecto non-bios

- **Otros ficheros que pueden aparecer en el árbol de proyectos:**

  Codigofuente.c: incluye el código fuente, es cambiado para cada programa.

  DSP281x_Adc.c: controla el convertidor analógico digital

  DSP281x_CpuTimers.c: configura los timers de la cpu y su frecuencia de trabajo.

  DSP281x_Ev.c: Controla el Event manager

  DSP281x_PieCtrl.c y DSPx_PieVect.c: ambos archivos incluyen rutinas para configurar y gestionar los vectores de interrupción y el PIE.

  DSP281x_CodeStartBranch.asm: Deshabilita el perro guardián.

1.9.4.- Diagrama de flujo de un programa ejemplo.

Todos los programas siguen el mismo patrón de diseño, o alguno muy parecido, quedando estos pasos reflejados en el siguiente diagrama de flujo (Figura 9.4)

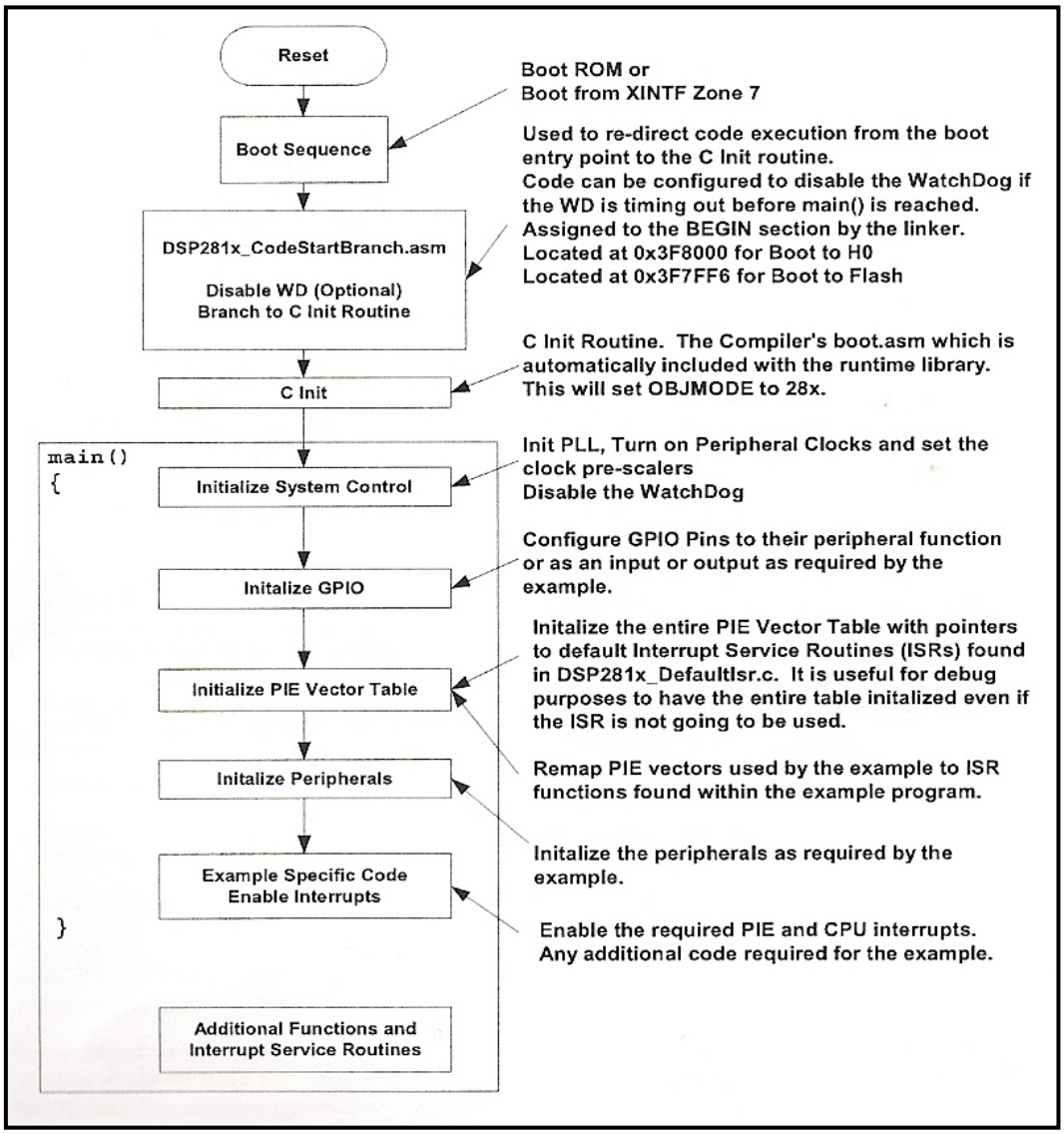


Figura9.4: Flujograma básico de un proyecto

Tras hacer un reset el DSP, ejecuta la secuencia de arranque. En el siguiente paso es donde se desconecta el perro guardián. A continuación, se ejecuta la rutina de inicio de un programa en C, que básicamente ejecuta el código fuente hasta la línea main().

De aquí en adelante es donde aparece el código principal de la función main(), comenzando por inicializar el sistema, luego configurar los puertos GPIO necesarios, se declaran los vectores de interrupción, se inicializan los periféricos y se escribe el código especifico de la aplicación a ejecutar.

Tras la función main() será necesario declarar las rutinas que sean necesarias.(ej. ISR).

1.9.5. Pasos a seguir para incorporar los ficheros de ayuda a la programación.

El primer paso para empezar a escribir código podría ser a partir de un ejemplo de los que proporciona el fabricante y modificarlo según sea conveniente. También se puede crear un proyecto desde el principio y empezar a escribir el código con la función main(), variables globales, rutinas, etc…

Una vez creado el proyecto, hay que escribir el código fuente con la función main() que es la que se ejecutara. En este fichero hay que incluir los ficheros .h que describen los periféricos y los registros que sirven para su configuración. Esto se hace escribiendo las siguientes líneas al principio.

#include "DSP281x_Device.h"

#include "DSP281x_Examples.h"

Hay que indicar al compilador donde se encuentran estos ficheros. Pulsando en el menú Project en la pestaña *build options*, entramos en el menú de *compiler* y seleccionamos la categoría *pre-processor*. Una vez dentro se añaden el path o ruta de los ficheros .h. La secuencia que habrá que escribir será:

C:\tidcs\c28\DSP281x\v111\DSP281x_common\include;

C:\tidcs\c28\DSP281x\v111\DSP281x_headers\include;

Es necesario añadir el fichero *DSP281x_GlobalVariableDefs.c*, al proyecto. Este fichero incluye la declaración de variables que son usadas para acceder al registros de los periféricos y la sección de datos que usara el enlazador. Se encuentra en la carpeta *.\DSP281x_headers\source.*

No hay que olvidar los ficheros de comandos .cmd que se han explicado anteriormente. Estos son *F2812_EzDSP_RAM_lnk.cmd* y *DSP281x_Header_nonBIOS.cmd.*

Para poder compilar el codigo .c hay que añadir un fichero de librería que se incluye con el entorno CCS. Este fichero es el *rts2800_ml.lib.*

Por último, si se emplea algún periférico o es necesario gestionar alguna interrupción es muy util incluir los ficheros .c que proporciona el fabricante relacionados con el periférico en cuestión y con la gestión de interrupciones.

Otra opción a la hora de añadir diferentes ficheros, tanto .c, .h y .cmd es la de buscar estos ficheros por los subdirectorios y copiarlos a la carpeta de trabajo y desde ahí añadirlos al proyecto.

Para la compilación existe la posibilidad de configurar el entorno CCS con diferentes opciones como activar el modo de memoria extensa, el chequeo del código y generación de warnings para poder mejora la programación.

Se recomienda seguir el esquema de programación de la función main() tal y como se indica en el apartado 9.4, empezando por la configuración del sistema, la configuración de los puertos GPIO y los periféricos. Si hay que emplear interrupciones es conveniente habilitarlas y después se incluye el código específico de la aplicación.

1.9.6.- Descripción de las variables y soporte a periféricos.

Esta tabla incluye las librerías de extensión .h y las funciones que tiene:

Fichero	Localizacion	Descripcion
DSP281x_Adc.h	DSP281x_headers\include	Estructura del convertidor AD
DSP281x_CpuTimers.h	DSP281x_headers\include	Registros de los timers de la cpu
DSP281x_DevEmu.h	DSP281x_headers\include	Emulación de los registros de definicion
DSP281x_ECan.h	DSP281x_headers\include	Registros de estructuras eCAN
DSP281x_Ev.h	DSP281x_headers\include	Resgistros y estructura del Event Manager
DSP281x_Gpio.h	DSP281x_headers\include	Proposito general de los GPIO
DSP281x_Mcbsp.h	DSP281x_headers\include	McBSP estructura de registro
DSP281x_PieCtrl.h	DSP281x_headers\include	Control de registros del PIE
DSP281x_PieVect.h	DSP281x_headers\include	Estructura de definicion de la tabla del PIE
DSP281x_Sci.h	DSP281x_headers\include	SCI estructura de registros.
DSP281x_Spi.h	DSP281x_headers\include	SPI estructura de registros.
DSP281x_SysCtrl.h	DSP281x_headers\include	Registros de las definiciones del sistema. Incluye el perro guardian, el PLL , el CSM y los registros del reloj
DSP281x_Xintf.h	DSP281x_headers\include	Interfaz de memoria externa.
DSP281x_XIntrupt.h	DSP281x_headers\include	Estructura de las interrupciones externas

Tabla 9.5: Librerías de periféricos.

Todas estas librerías generan los siguientes nombres de variables y su sección de datos. Todos ellos se recogen a continuación en la siguiente tabla:

Periferico	Direccion de inicio	Nombre de la estructura de variable
ADC	0x007100	AdcRegs
Code Security Module	0x000AE0	CsmRegs
Code Security Module Password Locations	0x3F7FF8-0x3F7FFF	CsmPwl
CPU Timer 0	0x000C00	CpuTimer0Regs
Device and Emulation Registers	0x000880	DevEmuRegs
eCAN	0x006000	ECanaRegs
eCAN Mail Boxes	0x006100	ECanaMboxes
eCAN Local Acceptance Masks	0x006040	ECanaLAMRegs
eCAN Message Object Time Stamps	0x006080	ECanaMOTSRegs
eCAN Message Object Time-Out	0x0060C0	ECanaMOTORegs
Event Manager A (EV-A)	0x007400	EvaRegs
Event Manager B (EV-B)	0x007500	EvbRegs
Flash & OTP Configuration Registers	0x000A80	FlashRegs
General Purpose I/O Data Registers	0x0070E0	GpioDataRegs
General Purpose MUX Registers	0x0070C0	GpioMuxRegs
McBSP Registers	0x007800	McbspaRegs
PIE Control	0x000CE0	PieCtrlRegs

Tabla 9.6: Estructura de registros de los periféricos.

El soporte para la configuración del PIE viene contenido en la tabla 9.7

Fichero	Localización	Descripción.
DSP281x_DefaultIsr.c	DSP281x_common\source	Es el servicio de interrupción de rutinas (ISRS) para la totalidad de la tabla del PIE. Usted puede elegir para probar una de las funciones o volver a su propio mapa ISR a la tabla de vectores PIE. Nota: Este archivo no se utiliza para proyectos con DSP / BIOS
DSP281x_DefaultIsr.h	DSP281x_common\include	Función prototipo para el registro ISRS. No puede ser usado con DSP / BIOS
DSP281x_PieVect.c	DSP281x_common\source	Crea una instancia de la tabla de vectores PIE. Este ejemplo se puede copiar en la tabla del PIE con el fin de inicializar con el valor por defecto del ISR.

Tabla 9.7: Descripción de ficheros, localización y descripción.

Rutinas ya creadas para la inicialización, configuración en un modo de funcionamiento para el Timer, el ADC, y otros periféricos, que pueden ser

aprovechadas en la programación se encuentran en los ficheros indicados en la Tabla 10.4.

Fichero	Descripción
DSP281x_GlobalPrototypes.h	Prototipos de funciones para el periférico, funciones específicas incluidas en estos archivos.
DSP281x_Adc.c	ADC, funciones específicas y macros.
DSP281x_CpuTimers.c	CPU-Timer, funciones específicas y macros.
DSP281x_ECan.c	Ampliación de la CAN de funciones específicas y macros.
DSP281x_Ev.c	Event Manager (EV), funciones específicas y macros.
DSP281x_Gpio.c	Entradas/salidas de propósito general (GPIO), funciones específicas y macros.
DSP281x_Mcbsp.c	McBSP, funciones específicas y macros.
DSP281x_PieCtrl.c	PIE, funciones específicas de control y macros.
DSP281x_Sci.c	SCI, funciones específicas y macros.
DSP281x_Spi.c	SPI, funciones específicas y macros.
DSP281x_SysCtrl.c	Sistema de control (watchdog, reloj, etc PLL), funciones específicas y macros.
DSP281x_Xintf.c	Interfaz de memoria externa (XINTF), funciones específicas y macros.
DSP281x_XIntrupt.c	Interrupciones exteriores, funciones específicas y macros.
DSP281x_CodeStartBranch.asm	Subdivisión para el inicio de la ejecución de código. Esto se utiliza para re-direccionar la ejecución de código cuando arranque la Flash, OTP o H0 SARAM. Una opción para desactivar el dispositivo de vigilancia ante la rutina de inicio C está incluido. Si está arrancando de XINTF Zona 7, el uso DSP281x_XintfBootReset.asm lugar.
DSP281x_usDelay.asm	Ensambla la función para insertar un tiempo de retardo en microsegundos. Esta función está a cargo del ciclo y debe ser ejecutado desde cero esperando al RAM.

Tabla 9.8: Ficheros y descripción.

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

2.-PLACA DE EXPANSIÓN PROPIA

En este apartado vamos a exponer el diseño de nuestra placa de expansión para la realización de los ejercicios prácticos de los apartados 3 y 4 que vienen a continuación.

Se justificara los cálculos realizados para que la placa tenga un correcto funcionamiento, también mostraremos las conexiones y disposición de los componentes, así como el trazado de las pistas.

2.1.- INTRODUCCIÓN

Para el diseño del esquemático y la placa de PCB, hemos utilizado el software producido por Cadence. Concretamente de la familia ORCAD los programas de Capture Cis y Layout plus en la versión V9.2.

La placa ha sido fabricada mediante fresadora y ha sido realizada a doble cara, tras realizar los cálculos teóricos del circuito, y comprobarlos por bloques en una protoboard.

2.2.- DISEÑO DE LAS PARTES

El DSP se conecta a nuestra placa de expansión mediante los conectores de pines P4,P5,P7,P8, ya mencionados en el apartado 1. 3, que podremos ver en las siguientes figuras:

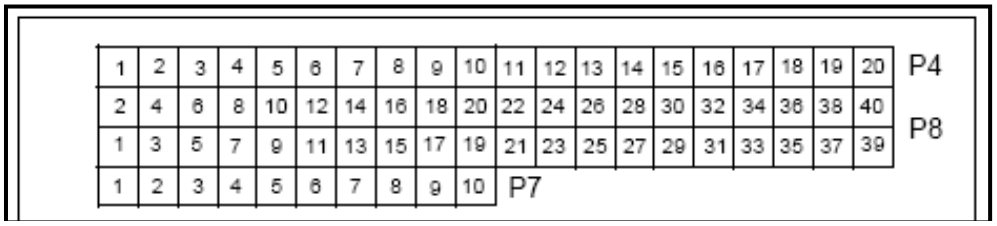


Figura 2.1: *conectores P4 ,P7,P8*

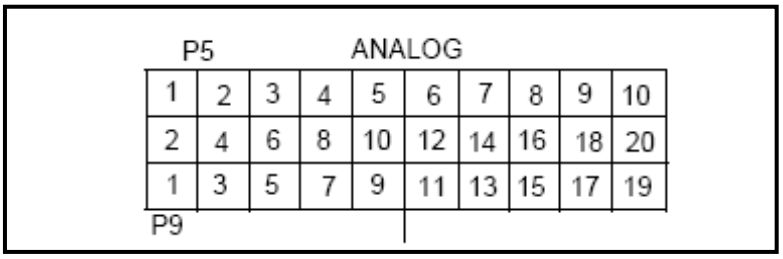


Figura 2.2: *conectores P5 yP9*

En la tabla 2.3 están indicadas las conexiones del DSP a los pines de los conectores de la placa, solo hemos indicado los necesarios para el funcionamiento de nuestras aplicaciones ,es decir las conexiones de leds, interruptores, pulsadores, entrada potenciómetro y altavoz.

Pines que se conectan a nuestra placa de expansión.

LED	**PUERTO DSP**	**CONECTOR**	**PIN**
1	B0	P8	30
2	B1	P8	31
3	B2	P8	32
4	B3	P8	33
5	B4	P8	34
6	B5	P8	35
7	B6	P4	13
8	B7	P4	14
PULSADOR D1	D1	P7	4
PULSADOR D6	D6	P7	8
INTERRUPTORES	**----------**	**----------**	**----------**
1	B8	P8	36
2	B9	P4	11
3	B10	P4	12
4	B11	P4	15
5	B12	P4	16
6	B13	P7	5
7	B14	P7	6
8	B15	P7	7
POTENCIOMETRO	A0	P9	2
ALTAVOZ	A6	P8	1
ALIMENTACION 5V	-	P8	1
GND	-	P8 P8 P4	40 39 20

Tabla 2.3: *Conexiones de los pines del DSP a los conectores de la placa.*

2.2.1.- Indicadores leds

Para la ejecución de los ejercicios de programación con el Dsp F2812 necesitamos 8 leds indicadores, son leds verdes que consumen unos 28 mA cada uno, por lo tanto demandarían al dsp 224 mA solo para los leds, esa demanda es demasiado por eso empleamos un buffer de corriente (74HCT244)

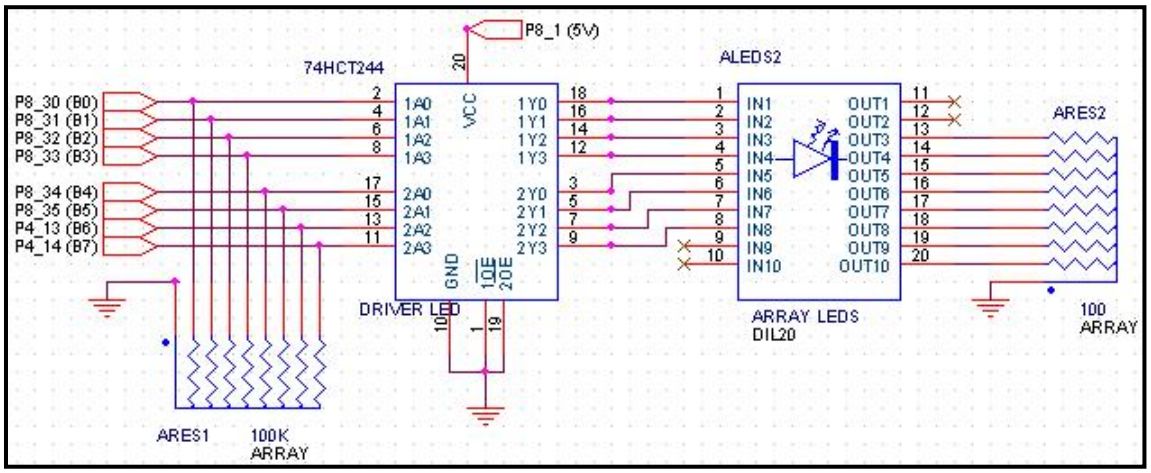


***Figura2.4**: Esquema de conexión y funcionamiento de los leds indicadores.*

Los drivers para alimentar los leds son 74HCT244 , los valores de alimentación recomendados están entre 4.5 y 5.5 V, siendo típico 5V, por tanto alimento a 5V (pin 1 del conector p8).
El integrado esta formado por dos grupos de cuatro entradas que se habilitan a nivel bajo con los pines 1OE y 2OE , mantengo siempre habilitado conectando a masa.

La caída de tensión en los leds verdes oscila entre 2.2 y 2.5V, y la corriente máxima es de 30mA. Por tanto el valor de corriente se ajusta con una resistencia de 100omios .

La alimentación de los driver 5V, descontando la tensión de los leds (en el peor de los casos 2.2 V) ,nos queda 2.8V con la resistencia de 100omios (95omios teniendo en cuenta 5%tolerancia),circula por los leds 29mA en el peor de los casos .

2.2.2 .-Interruptores

Con la disposición del esquema los interruptores dan un nivel alto 3,3V cuando el interruptor esta abierto ,la intensidad que entra al Dsp esta limitada por una resistencia de 100k para proteger el puerto con lo que obtenemos una corriente de entrada de 33uA.

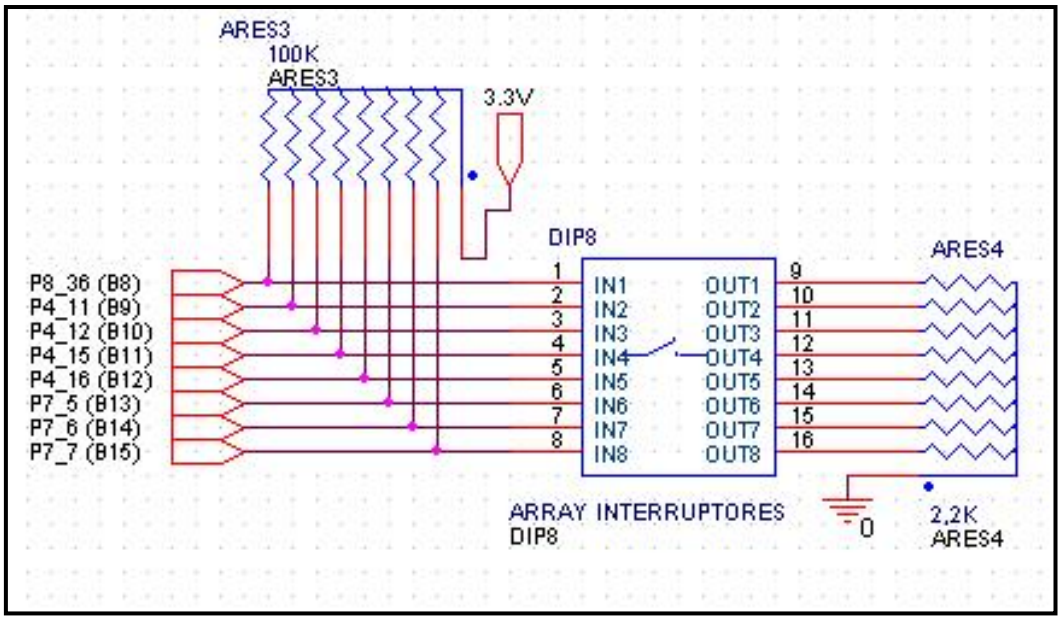


***Figura2.5**: Esquema de conexión y funcionamiento de los interruptores.*

2.2.3.- Pulsadores

Según esta disposición los pulsadores generan nivel alto (3,3V) cuando están sin pulsar.Tienen una resistencia de 100k que protege la entrada al Dsp ,para que la corriente de entrada sea 33uA.

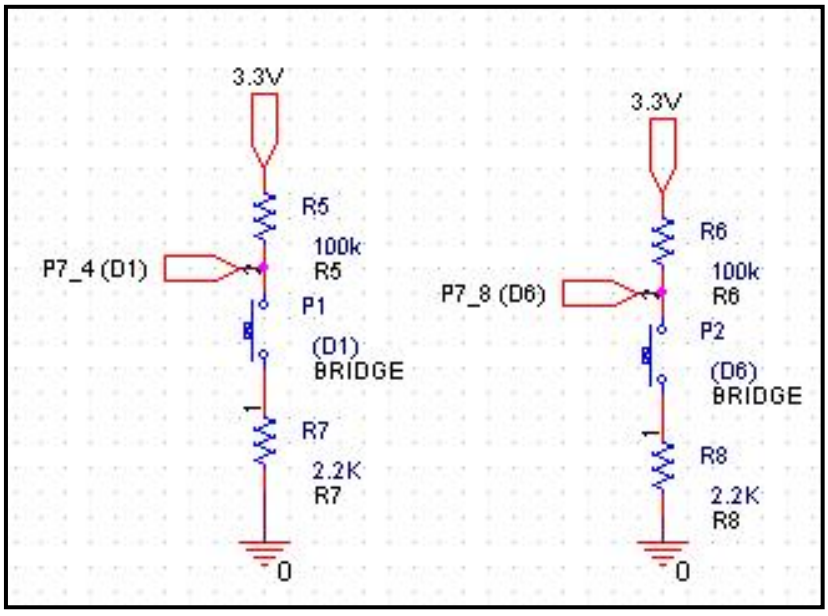


***Figura2.6**: Esquema de conexión y funcionamiento de los pulsadores.*

2.2.4.-Zumbador

Utilizamos un zumbador magnético de 12mm (cem_1203), alimentado a 3,3V.

Al ser un zumbador magnético, esta provisto de una bobina, por tanto utilizamos un diodo (1N4148) para que la corriente pueda salir cuando el transistor conmute y quede abierto (transistor en corte).

El transistor es utilizado a modo de relé para abrir y cerrar el circuito a la frecuencia del sonido que queremos generar, transistor trabaja entre corte y saturación.

Utilizamos el transistor BC337, la corriente máxima por la base es 200mA.

La resistencia de 180 Ω ,es suficiente para proteger la base del transistor ,teniendo en cuenta la caída de 0,7 V entre base y emisor (3,3V -0,7V /180Ω = 14.4mA).

Si tenemos en cuenta que la corriente máxima de entrada para el Dsp es 20mA ,obtenemos suficiente protección.

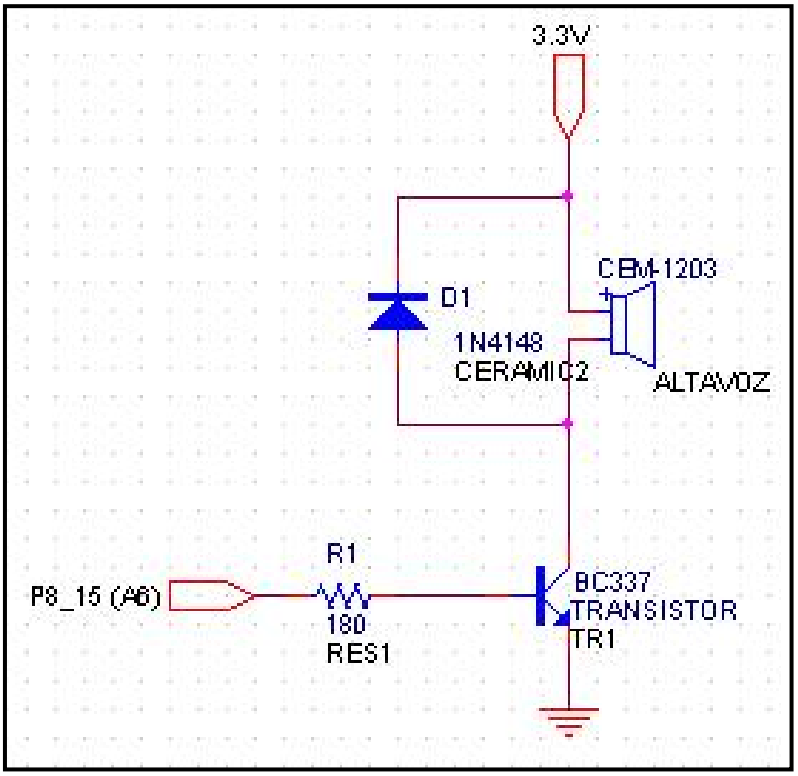


Figura2.7: *Esquema de conexión y funcionamiento del zumbador.*

2.2.5.- Potenciómetro.

Para adaptar el potenciómetro a la entrada del Dsp utilizamos un Amplificador operacional a modo de seguidor de tensión.

Hemos empleado el amplificador LM358, ya que se puede alimentar a 0 y 5V , y no necesita tensiones simétricas .

Debido al divisor de tensión a la entrada del operacional en el que utilizamos una resistencia de 1k la tensión máxima que puede generar el operacional es 3V , teniendo

en cuenta la resistencia del filtro paso bajo de la salida(150Ω) , la corriente máxima a la entrada del Dsp es 20 mA.

El filtro paso bajo a la entrada del Dsp sirve para que no se produzcan problemas al muestrear, nos aseguramos que se cumple el teorema del muestreo, con una frecuencia de corte del filtro de 1061hz.

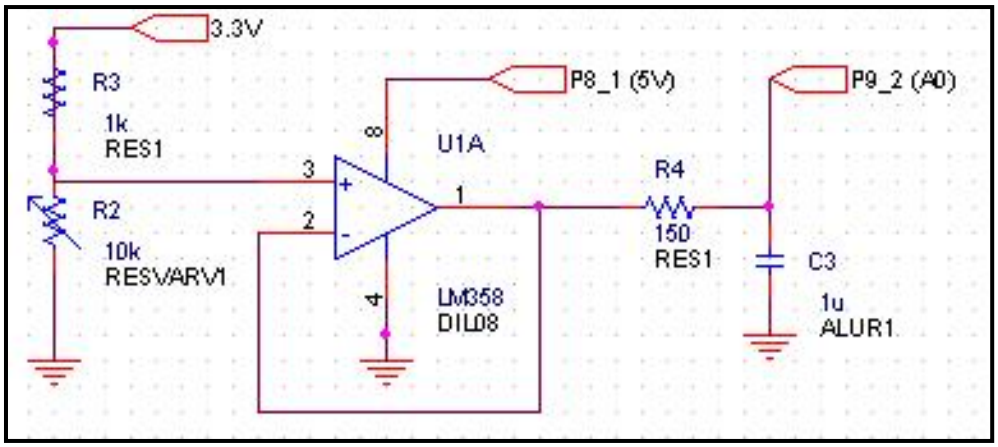


***Figura2.8**: Esquema de conexión y funcionamiento del potenciómetro con seguidor de tensión.*

2.2.6.- Regulador de tensión

Utilizamos un regulador de tensión con el cual obtenemos 3.3V a partir de la alimentación de 5V.
Para la alimentación se coge el pin 1 del conector p8 (5v)
El esquema seguido es el mismo que en el datasheet del estabilizador (LD1117v33) con un condensador cerámico de 100nf y uno electrolítico de 10 uf.

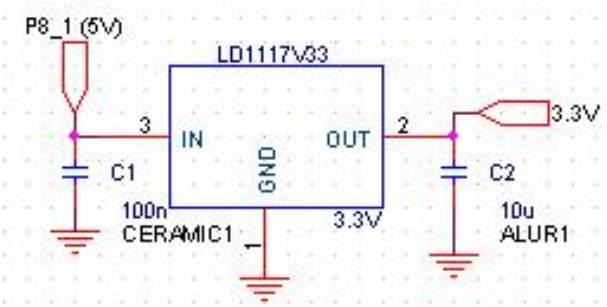


***Figura2.9**: Esquema de conexión y funcionamiento del regulador tensión 3.3V.*

2.3.- DISEÑO COMPLETO

A continuación mostraremos el diseño en conjunto de todas las partes, tal y como lo hemos utilizado para la creación de nuestra placa.

2.3.1.- Esquemático en Capture

En la siguiente figura se muestran las conexiones de todos los componentes de nuestro diseño.

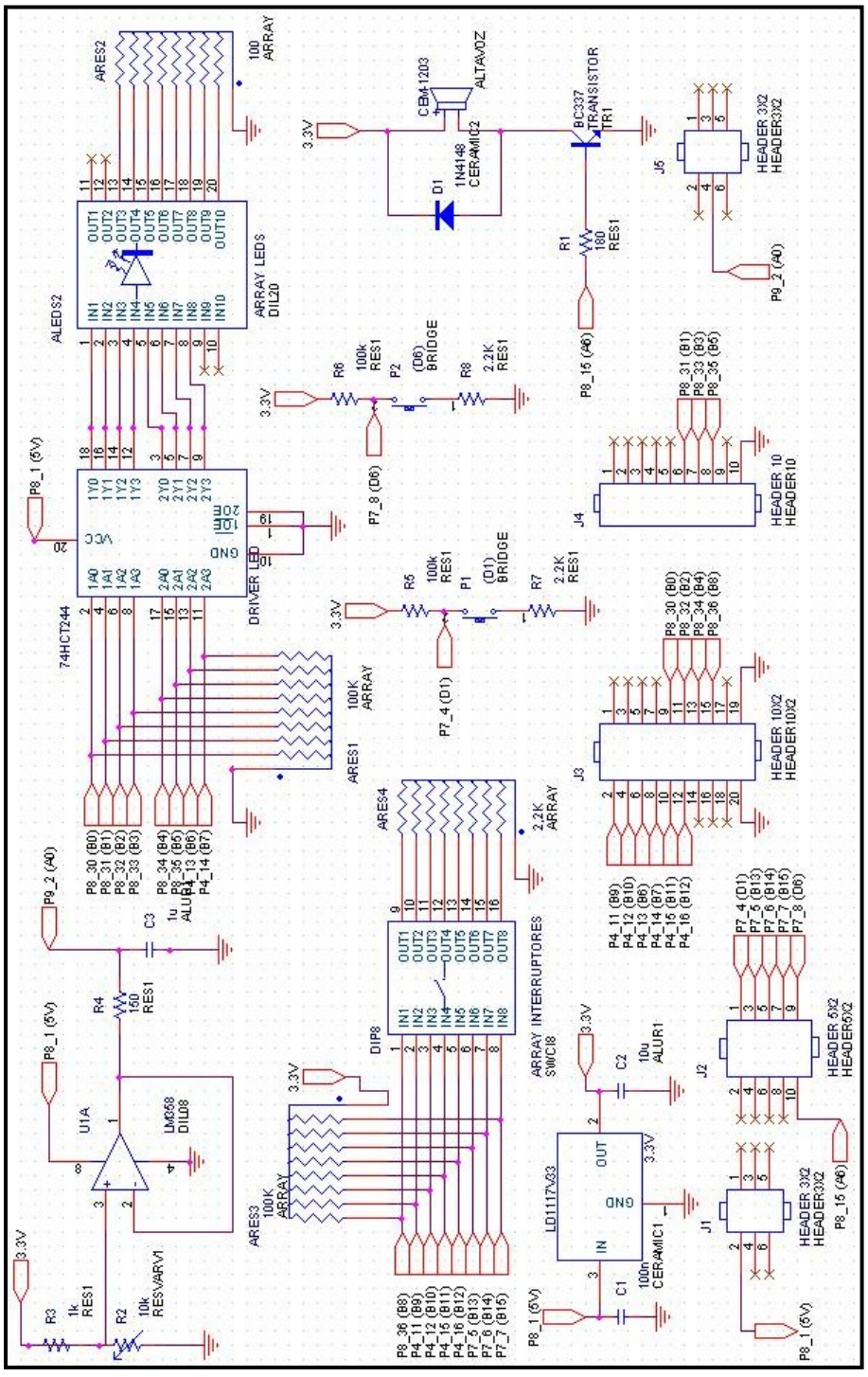


***Figura2.10**: Esquema de conexión y funcionamiento del conjunto.*

2.3.2.- Distribución de los componentes

Presentamos en este apartado la disposición de los componentes cara superior de la placa.

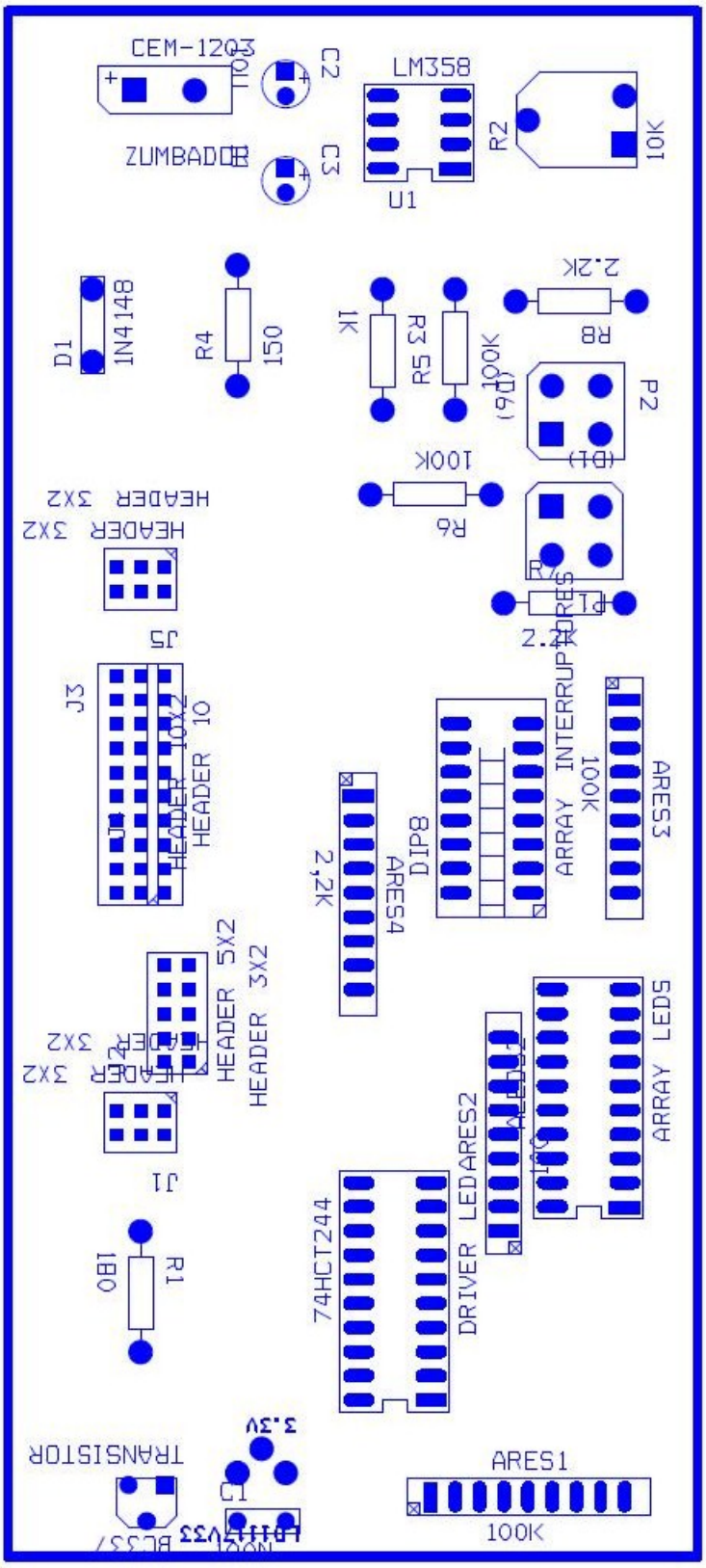


***Figura2.11**: Disposición de los componentes en la cara superior del PCB.*

2.3.3.- PCB cara superior

En este apartado mostramos el trazado de las pistas en la cara superior, donde están colocados los componentes.

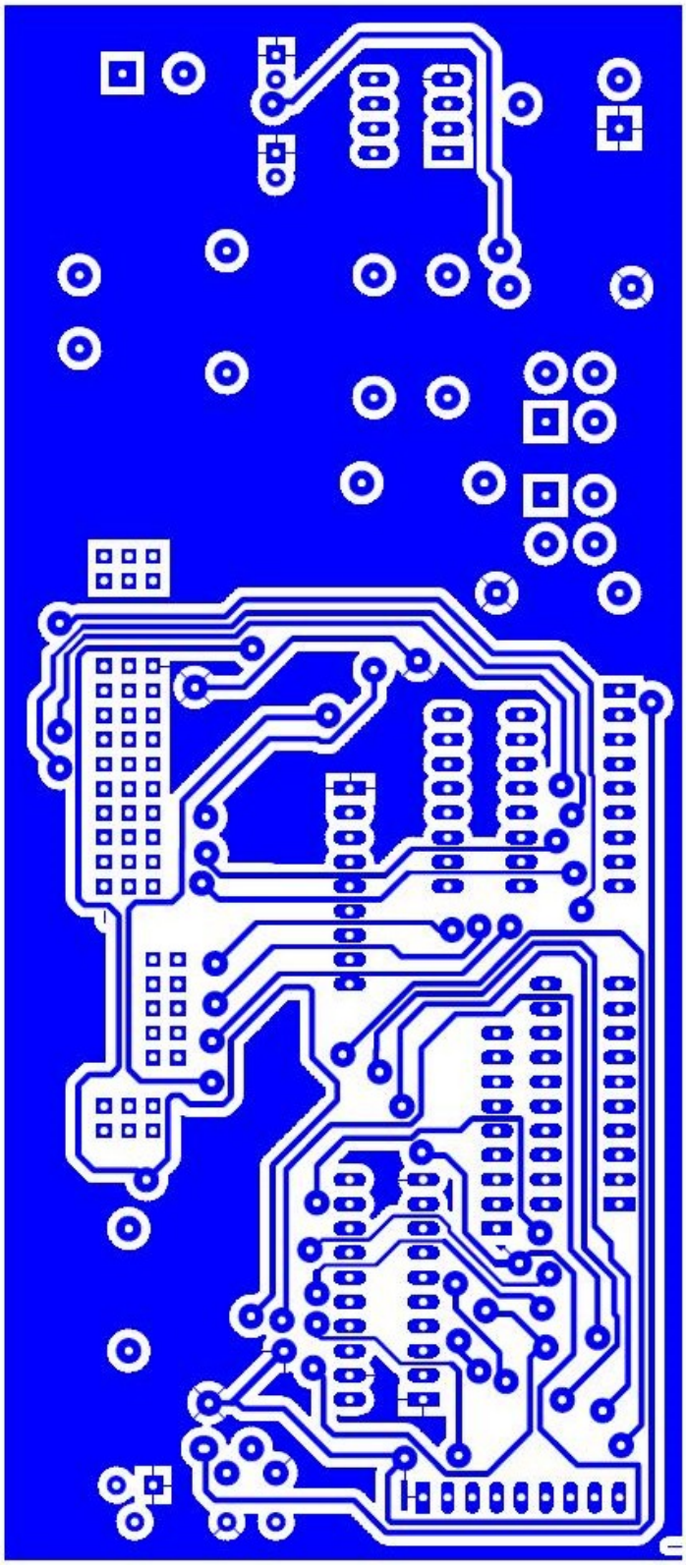


***Figura2.12**: Trazado de las pistas en la cara superior del PCB.*

2.3.4.-PCB cara inferior

En este apartado mostramos el trazado de las pistas en la cara superior, donde están los componentes.

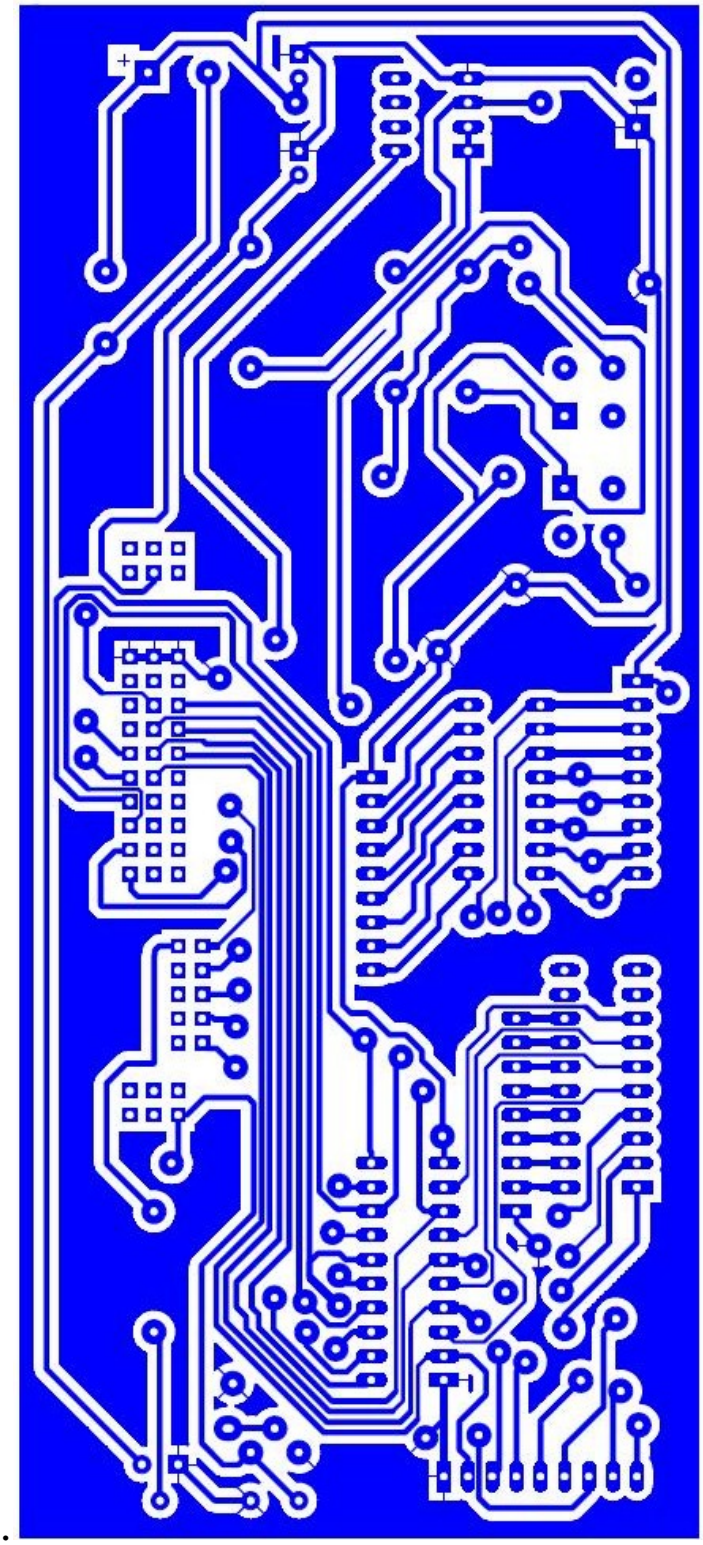


Figura2.1*3: Trazado de las pistas en la cara inferior del PCB.*

2.4.- PRESUPUESTO

El material utilizado se ha adquirido en el distribuidor electan (www.electan.com), a excepción de la placa de cobre que ha sido aportada por el departamento de tecnología electrónica de la UPCT.

1 x Zumbador 12mm	21.00%	1.60€
1 x Condensador electrolítico 10 uF. 25V	21.00%	0.14€
1 x Condensador cerámico 100 nF 100V	21.00%	0.14€
1 x Estabilizador de tension positiva 3.3V	21.00%	1.50€
1 x Pulsador CI ITT D6 Negro	21.00%	0.59€
1 x Pulsador CI ITT D6 Azul	21.00%	0.59€
1 x Mini interruptor DIP 8 pines	21.00%	1.07€
1 x Diodo de señales (1N914)	21.00%	0.04€
1 x Transistor BC337 NPN	21.00%	0.22€
1 x Resistencia 180 ohmios 1/4 W 5%	21.00%	0.03€
1 x Barra de 10 leds color verde 5,08 X 1,78 mm	21.00%	2.33€
2 x Resistencia array 8+1 100k ohmios	21.00%	0.48€
1 x Resistencia array 8+1 100 ohmios	21.00%	0.24€
1 x Buffer Schmitt tri-state octal	21.00%	0.26€
1 x Res. ajustable montaje horizontal ajuste vertical 10K ohmios	21.00%	0.45€
1 x Amplificador operacional	21.00%	0.41€
1 x Resistencia 1K ohmios 1/4 W 5%	21.00%	0.03€
1 x Resistencia 150 ohmios 1/4 W 5%	21.00%	0.03€
1 x Condensador electrolítico 1 uF. 50V	21.00%	0.14€
1 x Zócalo torneado 8 pines	21.00%	0.36€
2 x Zócalo torneado 20 pines	21.00%	1.12€
1 x Tira de pines doble hembra 40 pines paso 2,54 mm	21.00%	3.29€
1 x Tira de pines hembra 40 pines paso 2,54 mm	21.00%	1.31€
1 x Resistencia array 8+1 2k2 ohmios	21.00%	0.24€
2 x Resistencia 100K ohmios 1/4 W 5%	21.00%	0.06€
2 x Resistencia 2K2 ohmios 1/4 W 5%	21.00%	0.06€
	Subtotal:	16.73€
	Mensajería 48/72h Peninsula y Baleares (Solo pago con tarjeta o transferencia):	3.50€
	IVA 21%:	3.51€
	Iva Transporte 21%:	0.74€
	Total:	**24.48€**

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

3.-APLICACIONES SENCILLAS PARA EXPLICACION DE LOS RECURSOS DSP F2812

En este capítulo de aplicaciones básicas, procederemos a explicar los recursos del DSP F2812 mediante ejercicios sencillos, con poco código, de forma que se entiendan perfectamente y se puedan aplicar estos recursos para la programación de la Aplicación completa del apartado 4.

Los apartados de esta guía de aplicaciones básicas son:

3.1.- Manejo de Puertos Digitales.

Ejercicio A.- Encender los leds en una secuencia determinada.
Ejercicio B.- Encender leds empleando los interruptores.
Ejercicio C.- Utilizar pulsadores para activar una secuencia de luces.

3.2.- Manejo de interrupciones y timer.

Ejercicio A.- Encender los leds en una secuencia determinada, utilizando timer1 e interrupciones.
Ejercicio B.- Encender leds empleando los interruptores, timer 1 e interrupciones.
Ejercicio C.- Utilizar pulsadores para activar una secuencia de luces, empleando para ello el timer1 e interrupciones.

3.3.-Event Manager y generación de ondas.

Ejercicio A. Generación de una onda empleando timer1 e interrupciones
Ejercicio B.-Generación de una onda empleando el EVA.
Ejercicio C.-Generación de una onda PWM por comparación.

3.4.-Convertidor AD.

Ejercicio A.- Encender leds a partir de un determinado valor introducido mediante el potenciómetro.

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

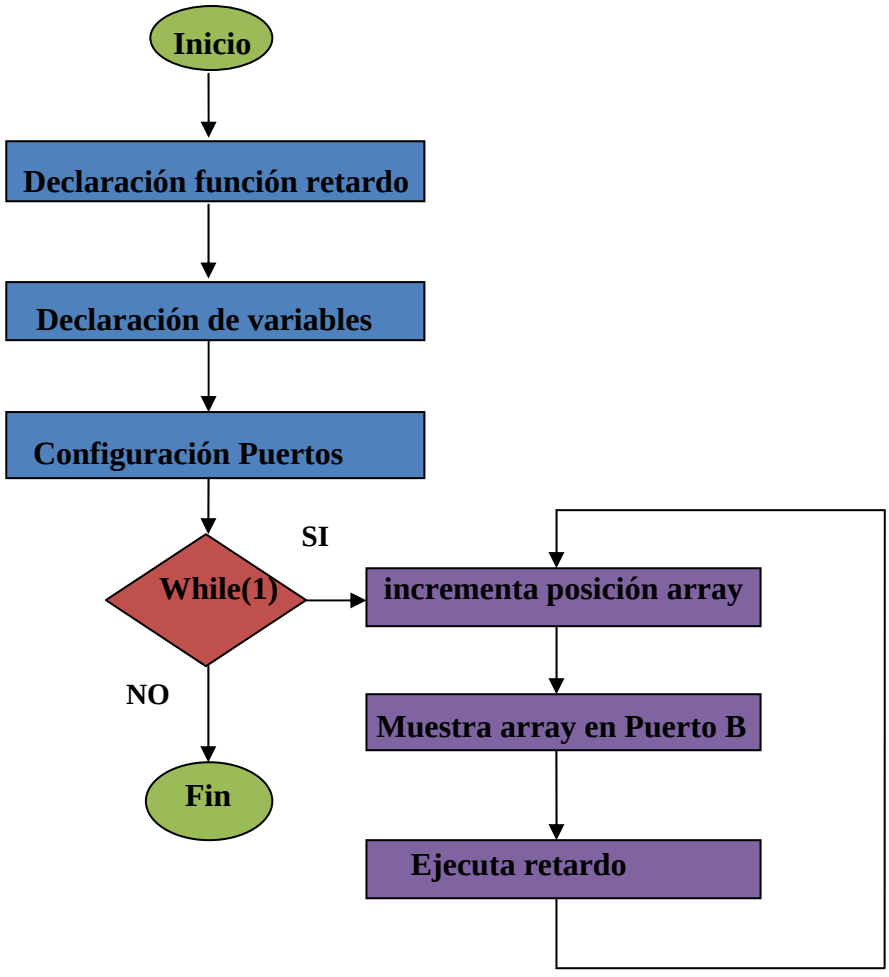
3.1.- MANEJO DE PUERTOS DIGITALES.

EJERCICIO A.

Utilizar los 8Led`s conectados al puerto GPIOB (bit0 a bit7), para mostrar una luz desplazándose de un lado a otro sin parar. De modo que se encienda los Led`s siguiendo la secuencia led1, led2, led3, led4, led5, led6, led7, led8 ascendiente y luego descendiente.

Pasos a seguir:

- Declarar función retardo.
- Declarar variables.
- Configurar puerto B (led`s) como puerto salida.
- Bucle While(1), recorrer el array con los valores del puerto B y hacer retardo.



Código:

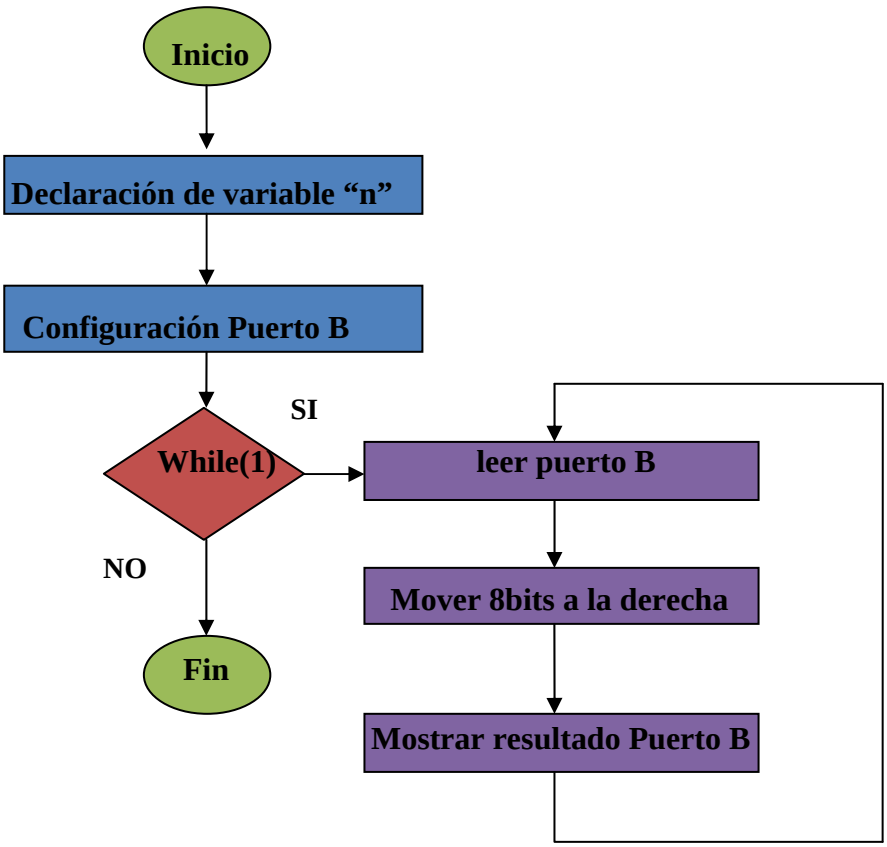
```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

void retardo(unsigned long valor){ //funcion retardo
unsigned long i;
for(i=0;i<valor;i++);}

int z;
int array[14]={1,2,4,8,16,32,64,128,64,32,16,8,4,2};

void main (void)
{
InitSysCtrl();

EALLOW;
GpioMuxRegs.GPBMUX.all=0x0000;//Asigna funcion de puerto datos al puerto B
GpioMuxRegs.GPBDIR.all=0x00FF;//Asigna funcion de salida de datos al puerto B
EDIS;

GpioDataRegs.GPBCLEAR.all =0x00FF;//pone a 0 el puerto B

while(1){

for(z=0;z<14;z++){
GpioDataRegs.GPBDAT.all = array[z];
retardo(2000000);
                  }
        }
}
```

EJERCICIO B.

Enciende cualquiera de los 8Led`s conectados al puerto GPIOB empleando los switches conectados al puerto GPIOB (bit8 a bit15).

Pasos a seguir:

- Declarar variables.
- Configurar puerto B
  - Bits(0-7) salida de datos
  - Bits(8-15)entrada de datos
- Bucle While(1):
  - Leer el puerto B
  - Mover el resultado a la derecha 8 bits (dividir entre 2^8=256)
  - Encender los leds (puerto B, bits 0-7)



Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

unsigned long n;

void main (void)
{
InitSysCtrl();
EALLOW;
GpioMuxRegs.GPBMUX.all=0x0000;//configuro puerto B como puerto de datos
GpioMuxRegs.GPBDIR.all=0x00FF;//puerto b:bits(0-7)salida y bits(8-15)entrada
EDIS;
GpioDataRegs.GPBCLEAR.all =0x00FF;

while(1){
n=GpioDataRegs.GPBDAT.all; //lee el puerto B
n=n/256; //divide entre 2^8=256 ,para mover el resultado 8bits a la derecha
GpioDataRegs.GPBDAT.all=n;//muestra por el puerto B
        }
}
```

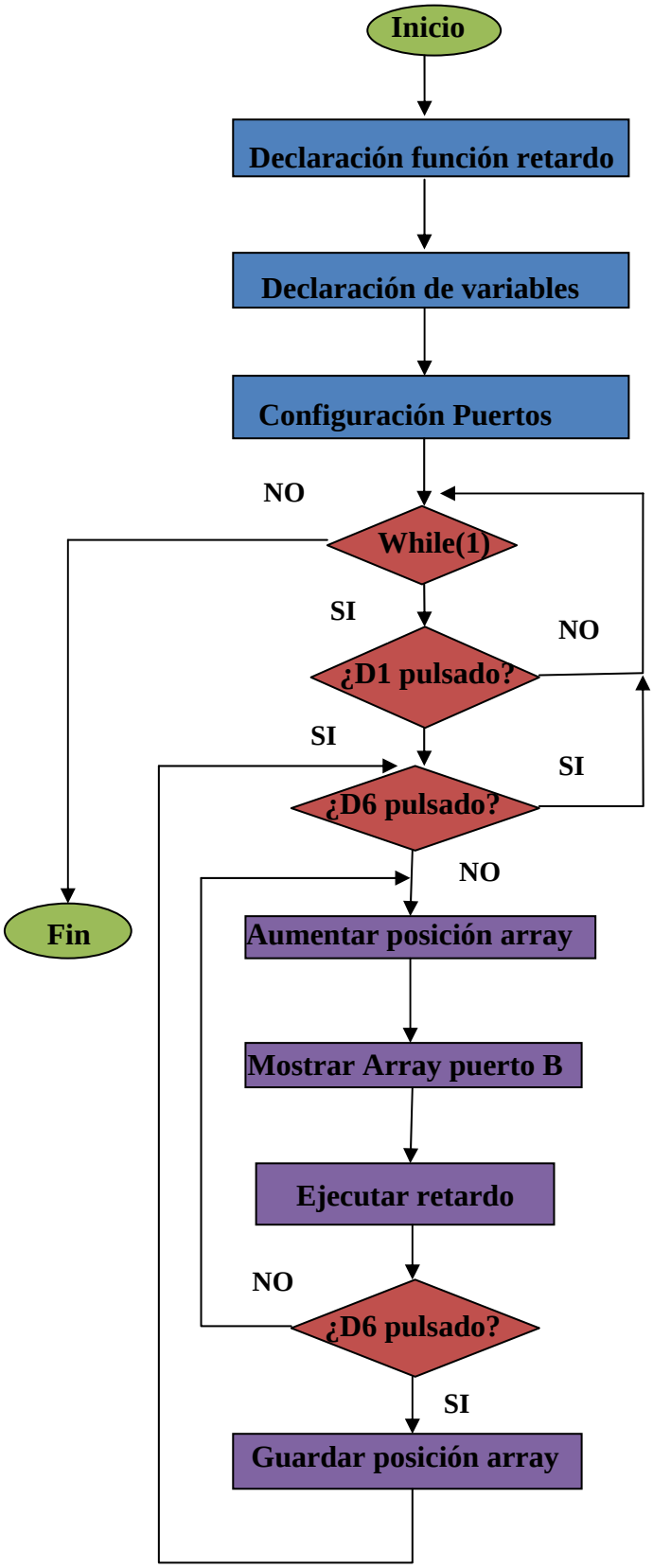
EJERCICIO C.

Con los dos pulsadores conectados al puerto D, arranca y para el movimiento de las luces (como en el programa A). Utilizar D1 para empezar el movimiento de las luces y D6 para parar.

-Al pulsar los pulsadores el nivel de entrada es "0", mientras que cuando no esta pulsados, el nivel de entrada es "1".

Pasos a seguir:

- Declarar variables.
- Configurar puertos
    - B, Bits(0-7) salida de datos
    - D, entrada de datos
- Bucle While(1):
    - Comprueba que se ha pulsado D1(Pulsador de marcha),para iniciar el recorrido del array.
    - Si se pulsa D6 (Pulsador paro) para y vuelve a estado inicial, guardando en la variable "y" la posición del array.


Inicio
Declaración función retardo
Declaración de variables
Configuración Puertos
NO
While(1)
SI
NO
¿D1 pulsado?
SI
SI
¿D6 pulsado?
NO
Fin
Aumentar posición array
Mostrar Array puerto B
Ejecutar retardo
NO
¿D6 pulsado?
SI
Guardar posición array

Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

void retardo(unsigned long valor)
{
    unsigned long i;
    for(i=0;i<valor;i++);
}
int y=0; //guardar la posicion del array de luces
int e;//pulsador D6
int i;//pulsador D1
int z;//recorrer el array
int array[14]={1,2,4,8,16,32,64,128,64,32,16,8,4,2};


void main (void)
{
InitSysCtrl();
EALLOW;
GpioMuxRegs.GPBMUX.all=0x0000;//configura la puerto B como funcion datos
GpioMuxRegs.GPBDIR.all=0x00FF;//configura 8 primeros bits puerto B como salida
EDIS;
GpioDataRegs.GPBCLEAR.all =0x00FF; //pone a 0 el puerto B(8 primeros bits)
while(1){
        i=GpioDataRegs.GPDDAT.bit.GPIOD1 ; //almacena el valor de D1
             while(i==0)      //si se ha pulsado
          {
            e=GpioDataRegs.GPDDAT.bit.GPIOD6 ; //comprueba que se ha pulsado D6
            if(e==0){break;}    //si se ha pulsado D6 sale
            for(z=y;z<14;z++)  //si no se ha pulsado recorre el array
            {
                y=0;
                GpioDataRegs.GPBDAT.all = array[z]; //muestra array por el puertoB
                retardo(2000000);   //aplica retardo para que se aprecie combio luz
                e=GpioDataRegs.GPDDAT.bit.GPIOD6 ;
                if(e==0){y=z;break;}     //si se pulsa D6 sale


                }
           }
        }
}
```

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

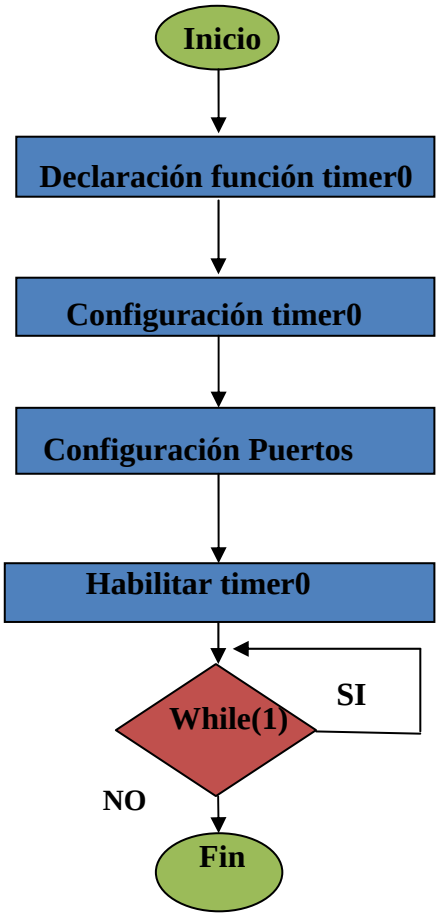
3.2.- MANEJO DE INTERRUPCIONES Y TIMER

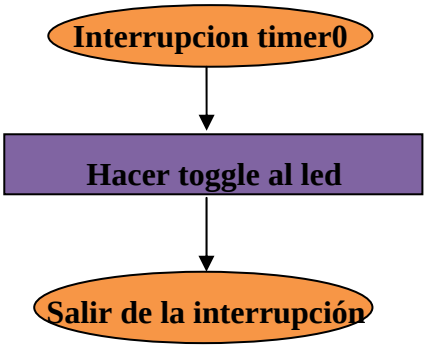
EJERCICIO A.-

Conseguir el parpadeo de un LED empleando el Timer0 y las interrupciones.

Pasos a seguir:

- Declarar función timer0.
- Inicializar tabla Pie y timer.
- Configurar timer0 mediante función ConfigCpuTimer().
- Configurar puerto B (led`s) como puerto salida.
- Configurar vector de interrupción.
- Habilitar línea int1 de interrupciones y la interrupción timer0 de la tabla pie.
- Función interrupción timer0.
    - Hacer toggle al led
    - Volver de la interrupción





Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

interrupt void cpu_timer0_isr(void);//funcion de inerrupción para timer0

void main (void)
{
     InitSysCtrl();
     InitPieCtrl();
     InitPieVectTable();  //inicializa tabla con vectores interrupccion
     InitCpuTimers();     //inicializa los timers
     ConfigCpuTimer(&CpuTimer0, 150, 1000000);//configura valores timer0
                    //frecuencia reloj 150Mhz y tiempo 1000000us=1s
     IER = 0x0000;//enable interrupciones a 0,desabilita interrupciones enmascaradas
     IFR = 0x0000;  //flag interrrupciones a 0

     EALLOW;
     GpioMuxRegs.GPBMUX.all=0x0000;
     GpioMuxRegs.GPBDIR.all=0x00FF;
     PieVectTable.TINT0 = &cpu_timer0_isr;//vectror interrupcion timer0
     EDIS;

     PieCtrlRegs.PIEIER1.bit.INTx7 = 1; //asignacion interrupccion timer0 (tabla PIE)
     IER |= 0x0001; //habilito linea de interrupcion int1
     EINT;
     ERTM;
     CpuTimer0Regs.TCR.bit.TSS = 0;//habilita  comienza a contar el timer0

   GpioDataRegs.GPBCLEAR.all =0x00FF;

   while(1){}
}

interrupt void cpu_timer0_isr(void)
{
   GpioDataRegs.GPBTOGGLE.bit.GPIOB0=1; //parapadea led con toggle cada interrupcion
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;//salida de la interrupción
}
```

EJERCICIO B.

Utilizar los 8Led`s conectados al puerto GPIOB (bit0 a bit7), para mostrar una luz desplazándose de un lado a otro sin parar. De modo que se encienda los Led`s siguiendo la secuencia led1, led2, led3, led4, led5, led6, led7, led8 ascendiente y luego descendiente.

-Generar la temporización que en el ejercicio 1.A se hizo con un bucle for, con el Timer0 de la CPU y las interrupciones .

-Utilizar la variable CpuTimer0.InterruptCount

Pasos a seguir:

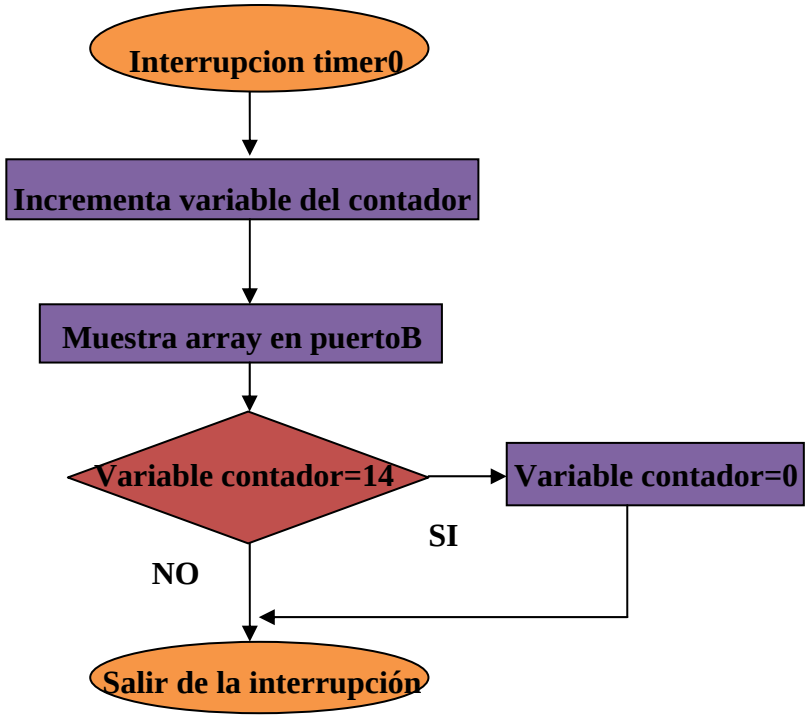
- Declarar función timer0.
- Inicializar tabla Pie y timer.
- Configurar timer0 mediante función ConfigCpuTimer().
    - Configura interrupción cada 100000us (0.1s)
- Configurar puerto B (led`s) como puerto salida.
- Configurar vector de interrupción.
- Habilitar línea int1 de interrupciones y la interrupción timer0 de la tabla pie.
- Función interrupción timer0.
    - Incrementa variable CpuTimer0.InterruptCount
    - Muestra por el puerto B
    - Sale de la interrupción

Explicación:

El programa es prácticamente el mismo que el anterior, el único cambio significativo es en la interrupción, donde en lugar de hacer parpadear un led utilizando toggle, se recorre un array para ir desplazando la iluminación de los leds siguiendo una secuencia.

El programa principal es prácticamente el mismo configuración de timer,puerto e interrupciones y a continuación bucle while(1).

Solo indicaremos el flujograma de la interrupción, ya que el programa principal es prácticamente idéntico.



Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

interrupt void cpu_timer0_isr(void);//funcion de inerrupción

int array[15]={0,1,2,4,8,16,32,64,128,64,32,16,8,4,2};
void main (void)
{
  InitSysCtrl();
  InitPieCtrl();
  InitPieVectTable(); //Inicializa tabla pie
  InitCpuTimers();  //Inicializa timer
  ConfigCpuTimer (&CpuTimer0, 150, 100000);//configura timer interrupcion cada 0.1s

   IER=0x0000;    //desabilita todas las lineas de interrupccion
   IFR=0x0000;     //quita todos flags lineas de interrupccion

   EALLOW;
   GpioMuxRegs.GPBMUX.all=0x0000; //configura puerto B
   GpioMuxRegs.GPBDIR.all=0x00FF;
   PieVectTable.TINT0=&cpu_timer0_isr;  //vector interrupcion
   EDIS;

   GpioDataRegs.GPBCLEAR.all=0x00FF;

   PieCtrlRegs.PIEIER1.bit.INTx7 = 1; //asigna interrupcion timer0 (tabla pie)
   IER |= 0x0001;  //habilita linea interrupccion int1
   EINT;
   ERTM;
   CpuTimer0Regs.TCR.bit.TSS = 0; //habilita timer0 (empieza a contar)
```

```
 while(1) {}
}


   interrupt void cpu_timer0_isr(void)
{
     CpuTimer0.InterruptCount++;   //variable del contador para recorrer el array
     GpioDataRegs.GPBDAT.all = array[CpuTimer0.InterruptCount];
      if(CpuTimer0.InterruptCount==14)
      {
         CpuTimer0.InterruptCount=0;
      }
  PieCtrlRegs.PIEACK.all=PIEACK_GROUP1; //terimina interrupcion
}
```

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

EJERCICIO C.

Con los dos pulsadores conectados al puerto D arranca y para el movimiento de las luces (como en el programa A). Utilizar D1 para empezar el movimiento de las luces y D6 para parar.

-Utilizar Timer0 e interrupciones en vez de una función de retardo mediante bucle for.

Pasos a seguir:

- Declarar función timer0.
- Inicializar tabla Pie y timer.
- Configurar timer0 mediante función ConfigCpuTimer().
    - Configura interrupción cada 100000us (0.1s)
- Configurar puerto B (led`s) como salida, y puerto D (interruptores) como entrada.
- Configurar vector de interrupción.
- Habilitar línea int1 de interrupciones y la interrupción timer0 de la tabla pie.
- Lee pulsador marcha variable “e” si ha sido pulsado habilita contador.
- Lee pulsador de paro variable “i” si ha sido pulsado deshabilita contador.
- Función interrupción timer0.
    - Incrementa variable CpuTimer0.InterruptCount
    - Muestra por el puerto B
    - Sale de la interrupción

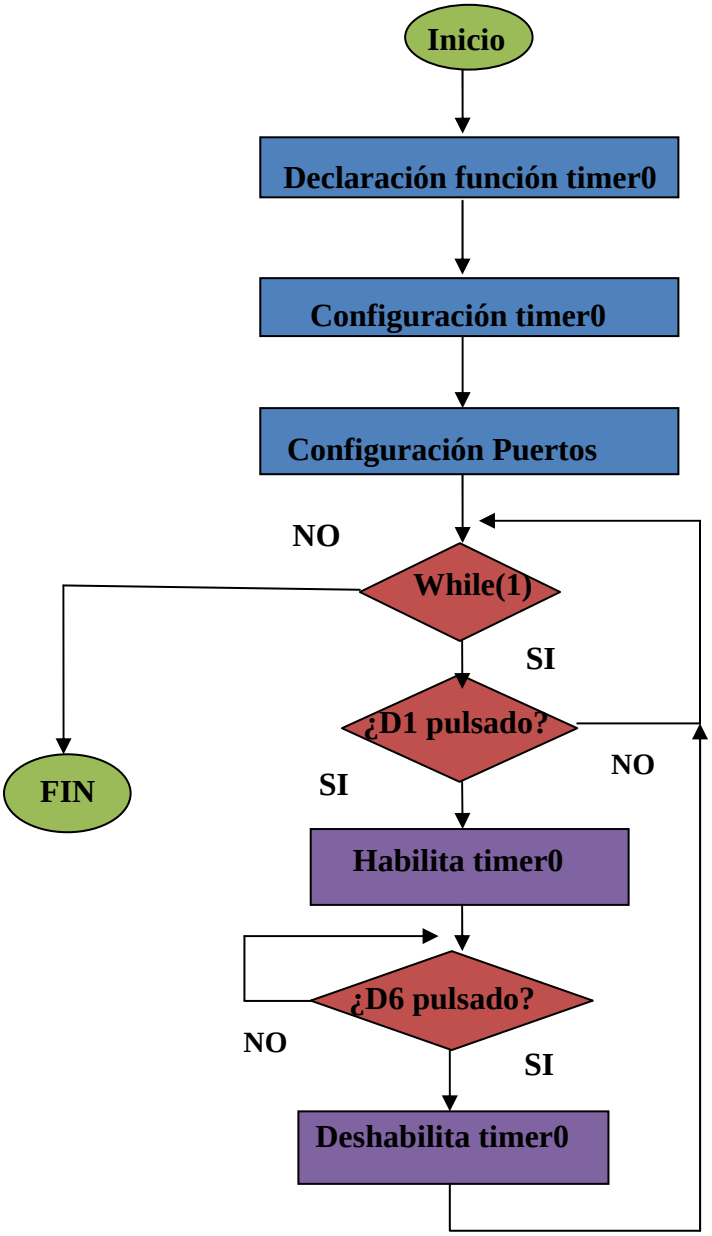
Explicación:

La función de interrupción de este programa es igual a la anterior (programa 2b),incrementa variable del contador y muestra siquiente posición del array.

La diferencia con el programa anterior esta en que este ejercicio requiere activar la secuencia de luces y desactivarla con los pulsadores conectados al puerto D (D1 marcha y D6 paro).

Para esto, en el programa principal While (1),se lee constantemente los pulsadores y cuando se pulsa D1 se habilita el contador y cuando se pulsa D6 se deshabilita.

Como la función de interrupción es la misma que en el ejercicio anterior se ha omitido, solo se muestra el flujograma principal.


Inicio
Declaración función timer0
Configuración timer0
Configuración Puertos
NO
While(1)
SI
¿D1 pulsado?
NO
SI
FIN
Habilita timer0
¿D6 pulsado?
NO
SI
Deshabilita timer0

Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

interrupt void cpu_timer0_isr(void);//funcion de inerrupción

int e;//pulsador puesta en marcha
int i;//pulsador paro
int array[15]={0,1,2,4,8,16,32,64,128,64,32,16,8,4,2};

void main (void)
{
   InitSysCtrl();
   InitPieCtrl();
   InitPieVectTable();
   InitCpuTimers();
   ConfigCpuTimer (&CpuTimer0, 150, 100000);//conf timer(interrupcion cada 0.1s)

   EALLOW;
   GpioMuxRegs.GPBMUX.all=0x0000; //conf puertoB salida y puerto D entrada
   GpioMuxRegs.GPBDIR.all=0x00FF;
   GpioMuxRegs.GPDMUX.all=0x0000;
   GpioMuxRegs.GPDDIR.all=0x0000;
   PieVectTable.TINT0=&cpu_timer0_isr;//vector interrupción timer0
   EDIS;

   GpioDataRegs.GPBCLEAR.all =0x00FF;
   GpioDataRegs.GPDCLEAR.all =0x00FF;

   IER=0x0000; //desabilita y quita flag todas lineas interrupcion
   IFR=0x0000;


   PieCtrlRegs.PIEIER1.bit.INTx7 = 1; //asocia interrupcion timer0 de la tabla pie
   IER |= 0x0001;  //habilita linea interrupcion int1
   EINT;
   ERTM;
   while(1)
   {
       i=GpioDataRegs.GPDDAT.bit.GPIOD1 ;//almacena valor Gpiod1
        while(i==0)  //si Gpiod1 ha sido pulsado
         {
             CpuTimer0Regs.TCR.bit.TSS = 0;//empieza a contar
            e=GpioDataRegs.GPDDAT.bit.GPIOD6 ; //guarda valor GPiod6
             if(e==0)    //si Gpiod6 ha sido pulsado
             {
                CpuTimer0Regs.TCR.bit.TSS = 1;//para de contar
                break;
             }
         }

   }

}
//funcion interrupcion igual que el ejercicio 2b anterior
//recorre el array con la variable del contador
interrupt void cpu_timer0_isr(void)
{
  CpuTimer0.InterruptCount++;
  GpioDataRegs.GPBDAT.all = array[CpuTimer0.InterruptCount];
      if(CpuTimer0.InterruptCount==14)
     {
      CpuTimer0.InterruptCount=0;
      }
  PieCtrlRegs.PIEACK.all=PIEACK_GROUP1;
}
```

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

3.3.- EVENT MANAGER Y GENERACIÓN DE ONDAS

EJERCICIO A.

Generar una onda cuadrada de frecuencia 400Hz y escucharla por el altavoz empleando el Timer0

-El altavoz esta conectado al pin GPIOA6/T1CMP_T1PWM

-Hay que utilizar el Timer0 y alternar la salida del pin a nivel alto y nivel bajo.

Pasos a seguir:

- Declarar función timer0.
- Inicializar tabla Pie y timer.
- Configurar timer0 mediante función ConfigCpuTimer().
    - Configura interrupción cada 1250us
- Configurar pin A6 como salida.
- Configurar vector de interrupción.
- Habilitar línea int1 de interrupciones y la interrupción timer0 de la tabla pie.
- Función interrupción timer0.
    - Toggle bit A6
    - Sale de la interrupción

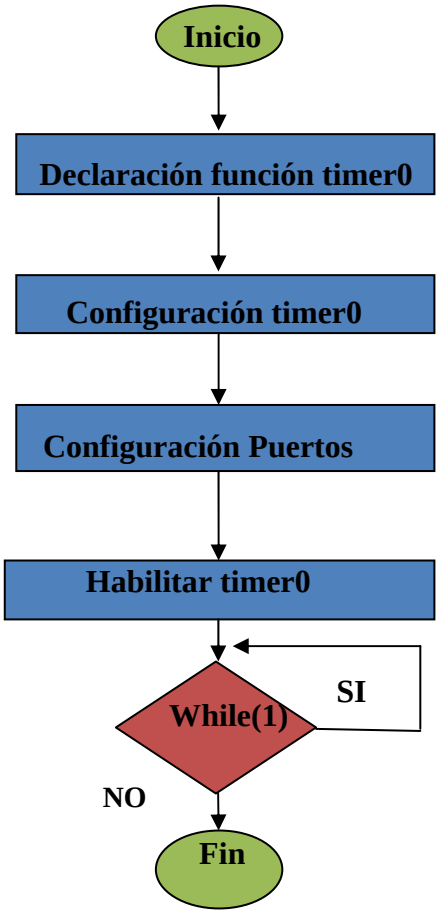
Explicación:

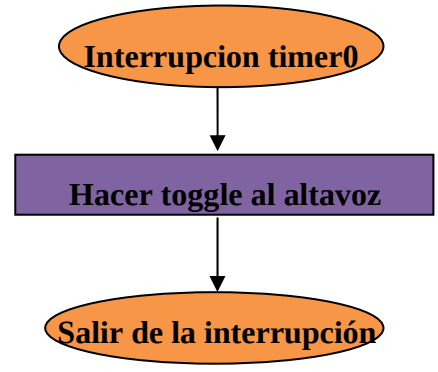
El programa genera una onda cuadrada de 400Hz para ello va alternando entre nivel alto y nivel bajo el pin A6 conectado al altavoz.

La interrupción tiene que saltar cada 1250us ,porque tiene que ser medio periodo de los 400Hz, medio periodo nivel alto y medio nivel bajo.

T= 1/400= 2.5ms

T/2=1.25ms=1250us


Inicio
Declaración función timer0
Configuración timer0
Configuración Puertos
Habilitar timer0
While(1)
SI
NO
Fin


Interrupcion timer0
Hacer toggle al altavoz
Salir de la interrupción

Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

interrupt void cpu_timer0_isr(void);//declaracion funcion interrupcion timer0

void main(void)
{
   InitSysCtrl();
   InitPieCtrl();
   InitPieVectTable();
   InitCpuTimers();
   ConfigCpuTimer(&CpuTimer0, 150, 1250);//cuenta la mitad del periodo de la señal

   EALLOW;
      PieVectTable.TINT0 = &cpu_timer0_isr; //vector interrupcion timer0
      GpioMuxRegs.GPAMUX.all=0x0000;
      GpioMuxRegs.GPADIR.bit.GPIOA6=1; //habilita pin A6(altavoz) como salida datos
   EDIS;

   PieCtrlRegs.PIEIER1.bit.INTx7 = 1; //asigna interrupcion tabla pie
   IER |= 0x0001;   //habilita int1
   EINT;
   ERTM;
   CpuTimer0Regs.TCR.bit.TSS = 0;      //habilita contador
   while(1){ }
}

interrupt void cpu_timer0_isr(void)
{
   GpioDataRegs.GPATOGGLE.bit.GPIOA6=1;//onda cuadrada que cambia de valor
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;//sale de la interrupcion
}
```

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

EJERCICIO B.

Generar una onda cuadrada de frecuencia 400Hz y escucharla por el altavoz empleando el Timer1 del EVA.

-El altavoz esta conectado al pin GPIOA6/T1CMP_T1PWM

-Hay que utilizar el Timer1 y alternar la salida del pin a nivel alto y nivel bajo.

Pasos a seguir:

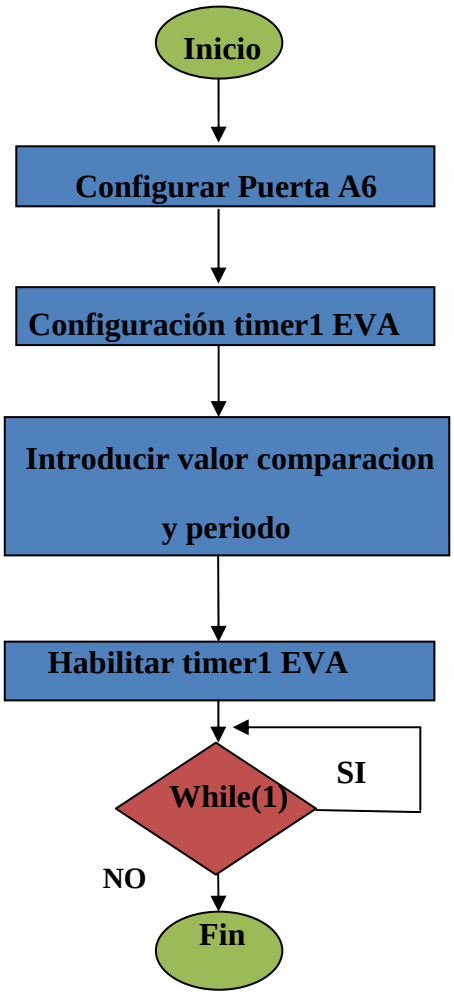
- Habilitar función primaria pin A6 (altavoz)
- Configurar timer1 EVA
    - Preescaler 1/128 ,por comparación , continuamente ascendente
- Introducir valor de comparación y periodo
    - Genera señal PWM 400Hz ciclo 50%(comparación = periodo/2)
- Habilitar timer1 EVA

Explicación:

Para generar la onda en este ejercicio usamos la función primaria del pin A6 que se encarga de generar onda PWM usando los timers del EVA y alternando entre nivel alto y bajo.

Configuramos la preescala a 1/128 ,y calculamos los tiempos de periodo y comparación para generar una onda de 400Hz,con un cilco del 50%.

- El reloj del timer1 del EVA funciona a la frecuencia Fhskl=75Mhz
- Temporización= Tosc x Preescala x cuenta
- Temporización= 1/400 = 2.5ms
- Cuenta= (Fos x Temporización)/Preescala
- Cuenta = (75Mhz x2.5ms)/128 =1464
- 1464 es el valor del periodo
- Valor de comparación= Valor periodo/2 =732



Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

void main(void)
{
      InitSysCtrl();

      EALLOW;
      GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=1;//habilita función primaria del pin
      EDIS;

      EvaRegs.GPTCONA.bit.TCMPOE =1;//habilita el registro de comparación
      EvaRegs.GPTCONA.bit.T1PIN =1; //activo a nivel alto
      EvaRegs.T1CON.all = 0x1702;//cofiguracion timer1,prescaler ect(0001011100000010)
       //habilita timer1 por comparacion, preescala 1/128, continuamente ascendente
      EvaRegs.T1CMPR=732;//valor comparacion
      EvaRegs.T1PR=1464;// valor de periodo
      EvaRegs.T1CON.bit.TENABLE=1; //habilita el timer1 del EVA
    while(1){  }
}
```

EJERCICIO C.

Generar una onda sinodal de 400Hz utilizando PWM de 50Khz.

-Hay que utilizar el Timer1 y que genere la onda PWM por comparación.

-Cada vez que se produzca una comparación, se activa una interrupción que recarga el registro con el nuevo valor de comparación.

-Se define los valores de la señal sinodal en una tabla y se va leyendo esa tabla cada vez que se produzca una interrupción por comparación.

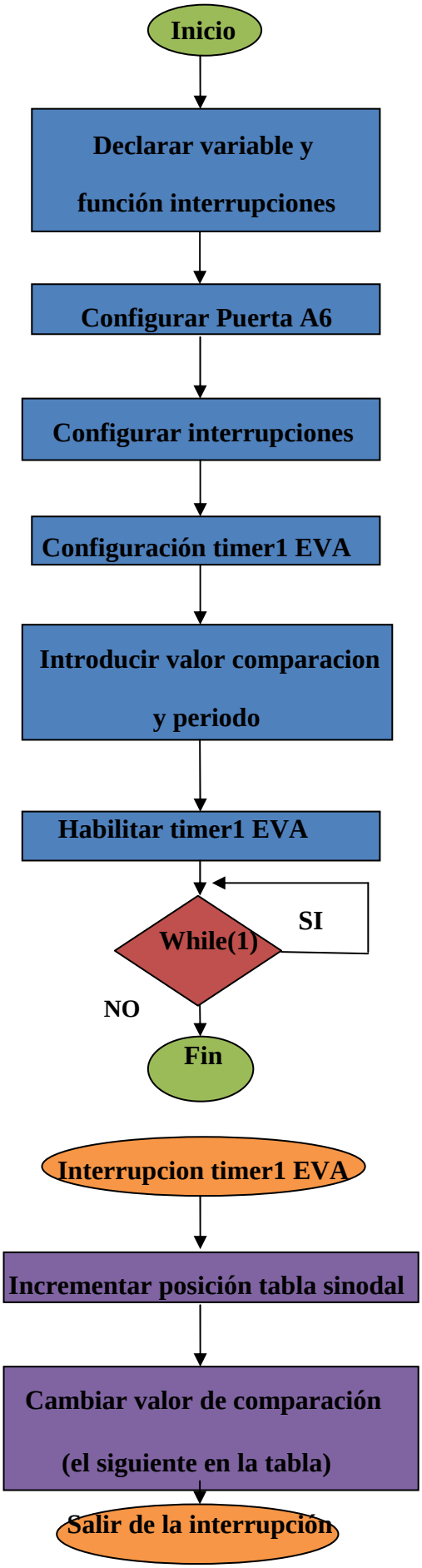
Pasos a seguir:

- Declarar función timer1
- Declarar variable z (recorrido tabla) y declarar tabla con valores sinoidal 400Hz.
- Inicializar tabla Pie .
- Configurar pin A6 como función primaria (PWM).
- Configurar vector de interrupción.
- Habilitar línea int1 de interrupciones y la interrupción timer0 de la tabla pie.
- Configurar timer1 EVA
- Introducir valor de comparación y periodo
- Función de interrupción
    - Incrementar posición array
    - Introducir valor de array(onda senoidal) en el valor de comparación
    - Salir de la interrupción
- **Explicación:**

Generamos una onda a partir de la comparación de una senoidal de 400Hz(valores de la tabla) y una triangular de 50Khz (timer1 del EVA).

Configuramos la preescala a 1/1 ,y calculamos los tiempos de periodo.

- El reloj del timer1 del EVA funciona a la frecuencia Fhskl=75Mhz
- Temporización= Tosc x Preescala x cuenta
- Temporización= 1/50Khz = 20us
- Cuenta= (Fos x Temporización)/Preescala
- Cuenta = (75Mhz x20us)/1 =1500
- 1500 es el valor del periodo


Inicio
Declarar variable y función interrupciones
Configurar Puerta A6
Configurar interrupciones
Configuración timer1 EVA
Introducir valor comparacion y periodo
Habilitar timer1 EVA
While(1)
SI
NO
Fin
Interrupcion timer1 EVA
Incrementar posición tabla sinodal
Cambiar valor de comparación (el siguiente en la tabla)
Salir de la interrupción

Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

interrupt void timer1(void);//funcion interrupcion timer1
//tabla con valores de la onda senoidal
unsigned int sine_table[512]={
750,741,732,723,714,705,695,686,677,668,659,650,640,631,622,613,604,595,586,577,5
,247,254,261,268,275,282,289,296,304,311,319,326,334,342,349,357,365,373,381,389,
1428,1432,1436,1440,1443,1447,1450,1454,1457,1460,1463,1465,1468,1471,1473,1476,1
,1012,1003,994,986,977,968,959,951,942,933,924,915,906,897,888,879,870,861,851,84
};
int z=0; //variable para recorrer la tabla

void main(void)
{
      InitSysCtrl();
      InitPieCtrl();
      InitPieVectTable();

      EALLOW;
      GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=1; //habilita funcion primaria
      PieVectTable.T1CINT = &timer1;//vector interupcion timer1
      EDIS;

      PieCtrlRegs.PIEIER2.bit.INTx5 = 1;//Asocia interupcion timer1 (tablas pie)

      IER=0x0000; //deshabilita todas las lineas de interrupcion
      IFR=0x0000;  //quita todos los flags de las lineas interrupcion

      IER |=0x0002; //habilito linea de interrupcion INT2
      EINT;
      ERTM;


      EvaRegs.GPTCONA.bit.TCMPOE =1;//habilita el registro de comparación
      EvaRegs.GPTCONA.bit.T1PIN =1; //activo a nivel alto
      EvaRegs.T1CON.all = 0x1002;// cofiguracion,prescaler ect
      EvaRegs.T1CMPR = sine_table[z];
      EvaRegs.T1PR=1500;// valor de periodo
      EvaRegs.T1CON.bit.TENABLE=1; //habilita el timer1 del EVA


   while(1){ }
}

interrupt void timer1(void){
z++;
  EvaRegs.T1CMPR = sine_table[z];//recorro los valores de la tabla
      if(z==511)
      {
      z=0;
      }
  PieCtrlRegs.PIEACK.all=PIEACK_GROUP2;//salgo de la interrupción timer1
}
```

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

3.4.- CONVERTIDOR A/D

EJERCICIO A.
Programar el convertidor A/D a una frecuencia de muestreo de 20Khz y que se enciendan los leds a partir de un valor umbral.

-El potenciómetro esta conectado a la entrada ADCINA0.

-Hay que programar el Timer1 del Event Manager para trabajar a 20KHz.

-Inicialmente los LEDs apagados y al variar el potenciómetro deben encenderse cuando pasen un umbral.

Pasos a seguir:

- Declarar valor Umbral (a partir del cual se activa leds)y Periodo(salta interrupción).
- Declarar función interrupción AD
- Declarar variable para lectura AD
- Configurar puertos (A función primaria y B salida datos)
- Inicializar tabla Pie .
- Configurar vector de interrupción.
- Configuración AD
- Configurar timer1 EVA
- Introducir valor de periodo
- Habilitar línea int1 de interrupciones y la interrupción AD de la tabla pie.
- While(1)
    - Comprobar si se pasa el valor umbral encender los leds si se cumple
- Función interrupción
    - Leer conversión AD y guradarla en una variable (Voltage_A0)
    - Resetear secuenciador y quitar flag

- **Explicación:**

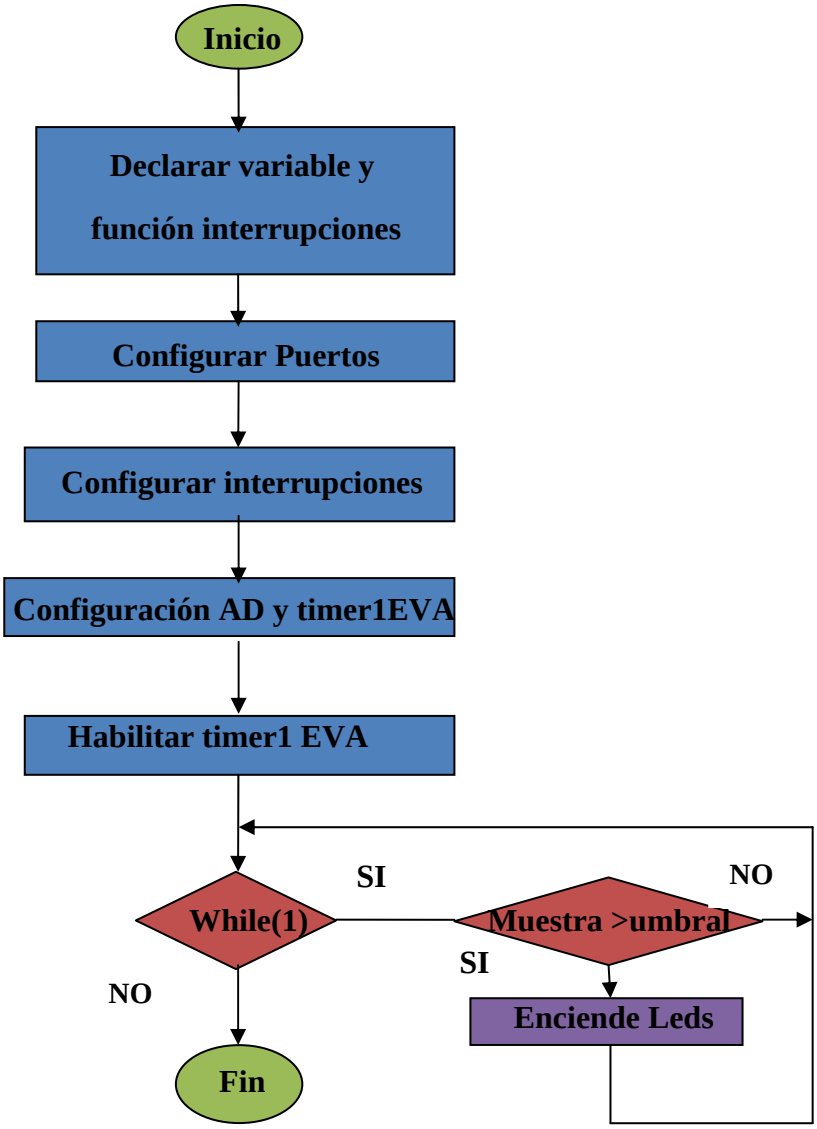
El programa muestrea el potenciómetro a una frecuencia de 20Khz automáticamente.

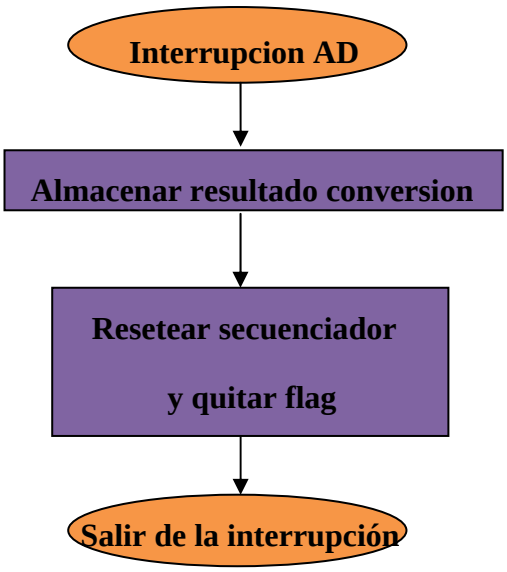
Solo si se pasa un determinado umbral se encienden los leds.

La explicación de la configuración del AD viene comentada en el código .

La configuración del timer1 es la siguiente :

- Configuramos la preescala a 1/1
- El reloj del timer1 del EVA funciona a la frecuencia Fhskl=75Mhz
- Temporización= Tosc x Preescala x cuenta
- Temporización= 1/20Khz = 50us
- Cuenta= (Fos x Temporización)/Preescala
- Cuenta = (75Mhz x50us)/1 =3750
- 3750 es el valor del periodo





Código:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

#define UMBRAL 2000  //valor a partir del cual se encienden los leds
#define T1PERIODO 3750//valor para frecuencia de 20kHz ,T=5x10^-5
//T=Tosc(hskl)x preescaler x cuenta
// cuenta=T/(Tosc x preescaler)
//cuenta =(5x10^-5)/(75x10^6)=3750

interrupt void ADC_Interrup(void);//vector interrupccion de la funcion  AD

int Voltage_A0;//variable donde se almacena la conversión

void main(void)
{
   InitSysCtrl();

   EALLOW;
   GpioMuxRegs.GPAMUX.all=0x00FF;// configuración puertos A funcion primaria
   GpioMuxRegs.GPBMUX.all=0x0000;//puerto B como salida de datos
   GpioMuxRegs.GPBDIR.all=0x00FF;
   EDIS;
   GpioDataRegs.GPBCLEAR.all=0x00FF;

   InitPieCtrl();
   InitPieVectTable();
```

```
   EALLOW;
   PieVectTable.ADCINT = &ADC_Interrup;//vectro interrupción del AD
   EDIS;

   //configuración del timer1 y del AD
   InitAdc();
   AdcRegs.ADCTRL1.bit.SEQ_CASC=0; //modo dual
   AdcRegs.ADCTRL1.bit.CONT_RUN=0; //para al final de la secuencia
   AdcRegs.ADCTRL1.bit.CPS=0;       //prescaler de la conversion 1/1
   AdcRegs.ADCMAXCONV.all =0x0000;  //(nº conversiones automaticas+1) //0+1
   AdcRegs.ADCCHSELSEQ1.bit.CONV00=0x0;//pone a 0 reg control secuencia conversion
   AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1=1; //conversiones son iniciadas por EVA(trigger)
   AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1=1;  //habilita el secuenciador 1
   AdcRegs.ADCTRL3.bit.ADCCLKPS=2;      //preescaler del reloj ADC
```

```
   EvaRegs.GPTCONA.bit.T1TOADC=2;//flag se activa cuando produce interrupcion
   EvaRegs.T1CON.all=0x1002; //configuracion timer1 EVA "0001 0000 0000 0010"
   //modo continuamente ascendente ,sin preescaler y funcionamiento por comparacion
   EvaRegs.T1PR=T1PERIODO;//tiempo de periodo

   PieCtrlRegs.PIEIER1.bit.INTx6=1; //asigna interrupion AD a la linea int1
   IER=0x0001;//habilita la linea int1 de interrupciones
   EINT; //habilita las  interrupciones
   ERTM;

   EvaRegs.T1CON.bit.TENABLE=1;//habilitación cuenta timer1
   while(1)
   {
     if(Voltage_A0>UMBRAL)//si esta por encima de unmbral enciende leds y si no, no
        GpioDataRegs.GPBDAT.all=0x00FF;
      else
         GpioDataRegs.GPBDAT.all=0x0000;
   }
}

interrupt void ADC_Interrup(void)
{  Voltage_A0 =AdcRegs.ADCRESULT0>>4;//almacenamiento de la conversión,desplaza 4bit
   AdcRegs.ADCTRL2.bit.RST_SEQ1=1;      //resetea el secuenciador
   AdcRegs.ADCST.bit.INT_SEQ1_CLR=1;    //quita flag
   PieCtrlRegs.PIEACK.bit.ACK1=1;       //salgo de la interrupcion
}
```

4.-APLICACIÓN DE USO FINAL

Caja de música protegida por código de apertura.

Esta es una aplicación que utiliza todos los recursos del DSP F2812 vistos en los ejercicios sencillos del punto 3

Consiste en una caja musical que cuando esta cerrada tiene una secuencia de luces y se abre pulsando un pulsador e introduciendo una clave por los interruptores, cuando se abre emite sonido por el altavoz, también tiene un sistema de alarma cuando esta cerrada.

4.1.- PLANTEAMIENTO DEL EJERCICIO

Se presentan 4 estados que facilitan el desarrollo del código para esta aplicación. Los estados son los siguientes:

Estado 0.-Caja cerrada

- Se encienden los leds siguiendo la secuencia del ejercicio **3.1.a**
  - Utilizar los 8Led`s conectados al puerto GPIOB (bit0 a bit7), para mostrar una luz desplazándose de un lado a otro sin parar.
- Al pulsar GPIOD1
  - Si la clave introducida en los interruptores (B8,B9,B10,B11) es correcta pasa al estado 1 (caja abierta).
    - Inicialmente el código de apertura es 0.
  - Si la clave es incorrecta se producirá un breve pitido por el altavoz y seguirá en el estado 0.
- El potenciómetro conectado al puerto A0 simula un sensor de movimiento ,cuando pase cierto umbral ,se activa la alarma. Si se activa la alarma:
  - Se encienden todos los leds del puerto B (bit0 a bit7)
  - Suena un sonido continuo (100Hz)
  - Para quitar la alarma pulsamos el botón(D6) reset durante 3s entonces:
    - Quita sonido y luces siguen la secuencia normal
    - Código de apertura vuelve a ser 0

Estado 1.-Caja abierta

- Parpadean 2 leds conectados al puerto (B6 y B7)
- Suena una melodía
  - Se genera variando la frecuencia de una señal generada por el puerto A6 y conectada al altavoz
- Si se pulsa D1 se vuelve al estado 0 (Caja cerrada)
- Si se pulsa D6 se pasa al estado2 (Ver y configurar código)

Estado 2.-Ver y configurar código

- Parpadean 3 leds conectados al puerto (B5 ,B6 ,B7)
- Muestra el código de apertura de la caja con 4 leds conectados al puerto(B0,B1,B2,B3)
- Suena una melodía igual que en el estado anterior
    - Se genera variando la frecuencia de una señal generada por el puerto A6 y conectada al altavoz
- Si se pulsa D6
    - Almacena el valor de los interruptores conectados al puerto (B8,B9,B10,B11) ,y esta será la nueva clave.
    - Pasa al estado 21

Estado 21.-Ver nuevo código

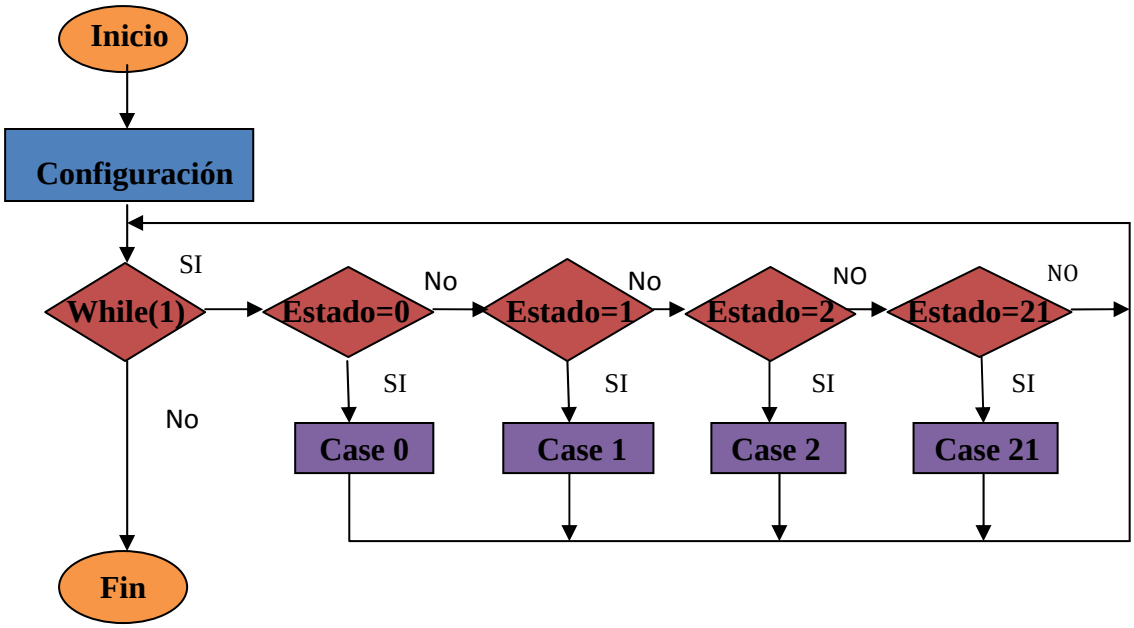
- Parpadean 3 leds conectados al puerto (B5 ,B6 ,B7)
- Muestra el código de apertura de la caja con 4 leds conectados al puerto(B0,B1,B2,B3) ,muestra el nuevo código ,después de que fuera cambiado en el estado anterior.
- Suena una melodía igual que en el estado anterior
- Si se pulsa D6 vuelve al estado 1 (Caja abierta)

Para hacer mas sencilla la explicación de esta Aplicación Completa, se ha dividido el flujograma y explicación en bloques.

4.2.- EXPLICACION DEL EJERCICIO

4.2.1.-Aplicación principal:

Esta compuesta en primer lugar por la configuración y a continuación un Case, donde se ejecutan unas instrucciones en función del estado en que se encuentre.
(0:caja cerrada - 1:caja abierta - 2: ver y configurar clave- 21:ver nueva clave)



A.-Bloque configuración:

- **Declaración de función interrupción**
  - Interrupción timer0 (CPUTimer0_Interrupt)
  - Interrupción AD (ADC_Interrupt)
- **Declaración de Variables**
  - **sonido_error=0** ,tiempo que debe estar activo el sonido error
  - **i=0** ,para recorrer el array de luces
  - **k=0** ,para recorrer el array de frecuencias
  - **codigo=0** ,codigo que abre la caja
  - **codigo_posible=0** ,codigo que se introduce por los interruptores
  - **mascara=0** ,mascara para el parpadeo de leds
  - **j=0** ,conteo para asegurar que se registra bien el estado del pulsador
  - **timestampCOCHE=0**, Para temporizar secuencia luces caja cerrada
  - **timestampERROR=0,** Para emitir el sonido de error
  - **resetCODIGO=0** ,para el conteo de 3s y hacer reset posterior
  - **DatoA0=0** ,almacena muestra AD , la tensión del potenciometro que indica manipulacion de la caja
  - **umbralA0=2000** ,umbral para activar alarma

- **alarmaON=0** ,**1.**alarma activada **0.**alarma desactivada
- **dummy=0** ,para temporizar 0.1s y cambiar a la siguiente posición del array de luces
- **led[14]={1,2,4,8,16,32,64,128,64,32,16,8,4,2}** ,secuencia de luces caja cerrada
- **frecuencia[15]={2219,1973,1776,1665,1480,1332,1184,1110, 1184,1332,1480,1665,1776,1973,2219} ,** melodía caja abierta
- **botonD1** ,registra pulsacion D1 apertura y cierre de caja
- **botonD6** ,registra pulsación D6 para pasar a estado manipulacion
- **estado=0** ,indica el estado en el que nos encontramos (**0.**caja cerrada **1.**caja abierta **2.** Ver y configurar código **21.**ver nuevo código)

- **Configuración de puertos**
  - Puerto A6 : salida de datos (altavoz)
  - Puerto B: salida de datos (led`s)
  - Puerto D1: entrada de datos (pulsador apertura y cierre)
  - Puerto D6: entrada de datos (pulsador de manipulación)

- **Configuración de interrupciones**
  - Inicializa tabla PIE
  - Asigna vector a la función de interrupción del timer0
  - Asigna vector a la función de interrupción del convertidor AD
  - Habilita grupo 6 de la tabla PIE (timer0,línea int1)
  - Habilita grupo 7 de la tabla PIE (convertidor AD, línea int1)
  - Habilita línea de interrupciones int1
- **Configuración del convertidor AD**
  - Modo dual
  - Modo no continuo (para al final de la secuencia)
  - Preescaler de la conversión 1/1
  - Numero de conversiones =1
  - Poner a 0 el registro de control de secuencia de conversión
  - Conversiones iniciadas por el EVA
  - Habilita secuenciador 1
  - Preescaler de reloj AD ¼
  - Conversión AD activada por el timer2 del EVA

- **Configuración timer1 EVA**
  - Funcionamiento por comparacion
  - Preescaler 1/128
  - Continuamente ascendente
  - Activo a nivel bajo

- **Configuracion timer2 EVA (disparo conversor AD automatico)**
  - Funcionamiento por comparacion
  - Preescaler 1/1
  - Continuamente ascendente
  - Configura periodo (3750,para el muestreo a 20Khz)
  - Habilitarlo para que empiece a contar

- **Configuración timer0**
  - Configurar timer para que salte interrupción cada 10us
  - Habilitarlo para que empiece a contar

B.-Case 0 (caja cerrada):

- Se comprueba si el valor que registra el AD (DatoA0) es mayor que el umbral que hemos definido, si es mayor se activa la alarma.
- Se comprueba si se ha pulsado D1
  - Si se pulsa se hace un bucle While ,un pequeño retardo para asegurar que se registra bien el estado del pulsador y evitar el efecto rebote.

    while(botonD1==0) { for(j=0;j<500000;j++)

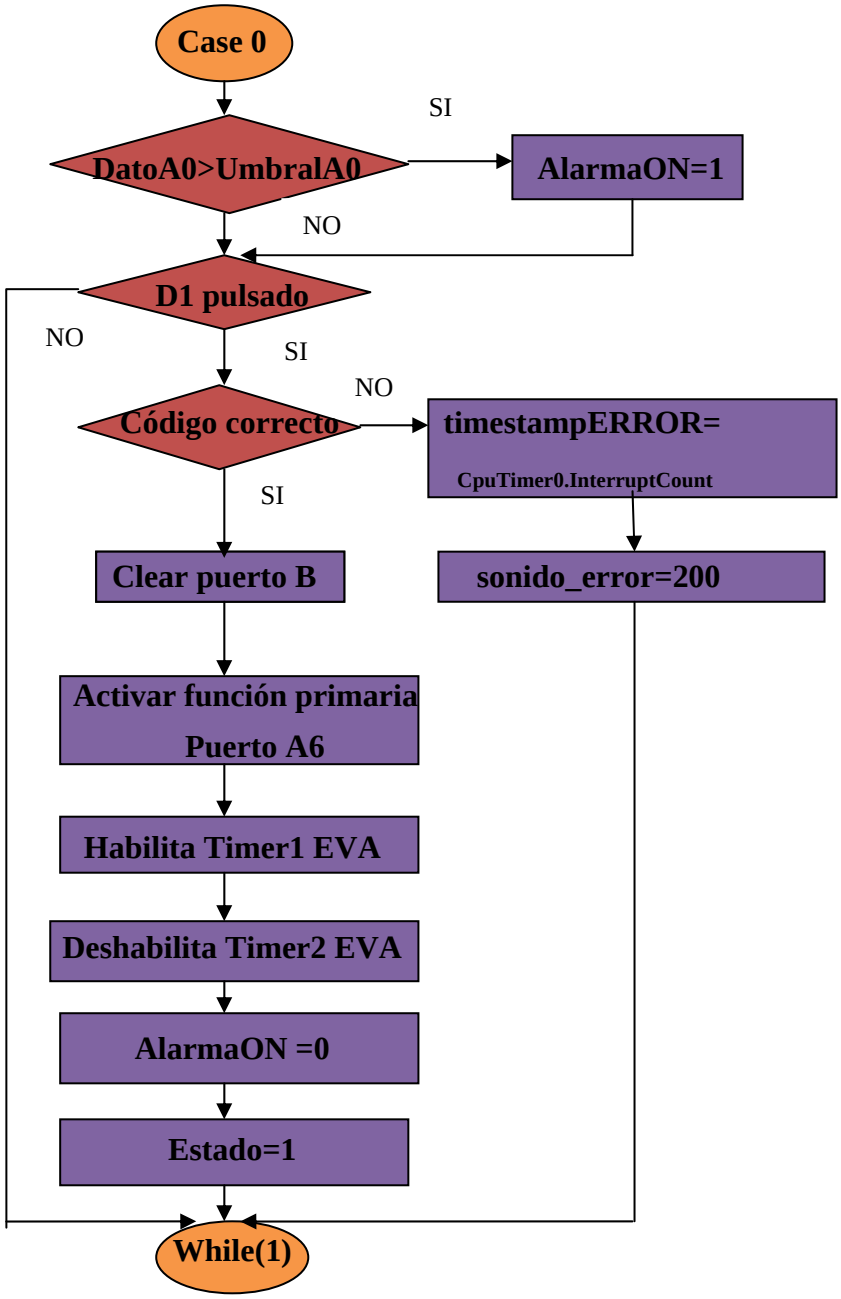
    botonD1=GpioDataRegs.GPDDAT.bit.GPIOD1; }

- Se comprueba el código introducido
  - En el puerto B ,los bits 15..8 corresponden a los pulsadores y los bits 7..0 corresponden a los leds.
  - Para quedarnos con los bits que nos interesa corremos los bits del puerto B ,8 posiciones a la derecha con la instrucción: **GpioDataRegs.GPBDAT.all>>8,** de este modo tenemos los 8bits mas significativos(interruptores) en la posición de los bits menos significativos (bit7…bit0).
  - Debido a las conexiones de los interruptores, cuando están cerrados dan un "1" logico y cuando están abiertos un "0",de modo que no se corresponden con los leds los cuales se encienden con un "1" lógico, asi que para que se correspondan se hace un" toggle" a los bits.
  - Para quedarnos solo con los 4 primeros bits ,que son los que nos interesa hacemos una operación **"& 00001111**"
  - Para invertir el resultado, al numero **15** (**0b1111**) le restamos el numero obtenido en el punto anterior
  - Con estos pasos tenemos "el toggle" de los 4bits de los interruptores que nos interesa y lo almacenamos en la variable **codigo_posible**

- Si el código es incorrecto
  - Realizamos un breve pitido con las siguientes instrucciones, explicaremos el funcionamiento cuando expliquemos la rutina de interrupción del timer0.
  - **timestampERROR=CpuTimer0.InterruptCount**
  - **sonido_error=200**

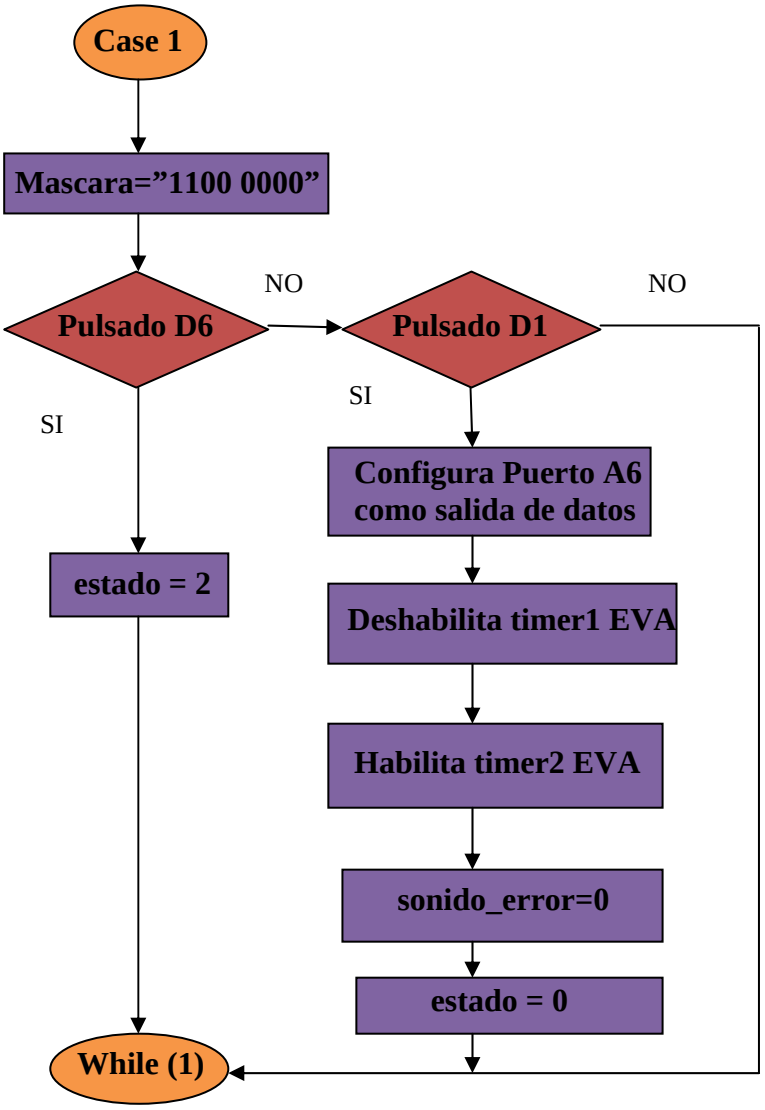
- Si el código es correcto
  - Hacer clear en el puerto B (leds)
  - Activamos la función primaria del puerto A6 conectado al altavoz y que generara la melodía de caja abierta (**ver explicación función timer0**).
  - Habilitar timer1 del EVA, utilizado para generar la melodía
  - Deshabilitar timer2 EVA,para que no tome muestras AD mientras esta en estado de caja abierta
  - Desactivar la alarma(AlarmaON=0)
  - Pasar a estado caja abierta (estado=1)

- Vuelve al bucle while(1)



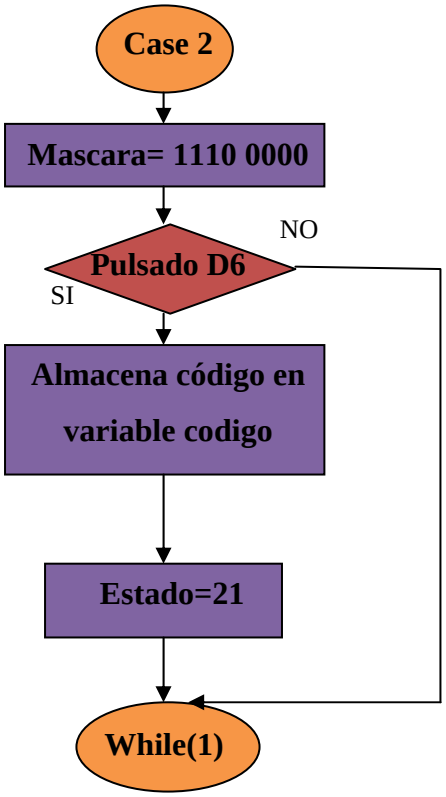
C.-Case 1 (caja abierta):

- Se da el valor "**1100 0000**" a la variable mascara ,para que solo parpadeen los leds (7 y 6) conectados al puerto B lo cual indicara el estado de caja abierta, para comprenderlo mejor **ver la explicación de la función del timer0**
- Comprobamos si se ha pulsado el pulsador conectado a D6(para evitar problemas utilizamos un bucle while explicado anteriormente).
- Si se ha pulsado D6 pasamos a estado2 (intro_pass)
- Si no se ha pulsado D6,entonces comprobamos si se ha pulsado D1
- Si no se ha pulsado ni D6 ni D1 volvemos al While(1)
- Si se ha pulsado D1 volvemos al estado 0 (caja cerrada) para ello:
  - Deshabilitamos función primaria de A6 se queda en salida datos
  - Deshabilitamos timer1 EVA
  - Habilitamos timer2 EVA
  - sonido error=0
  - estado=0
- Se vuelve al bucle while(1)



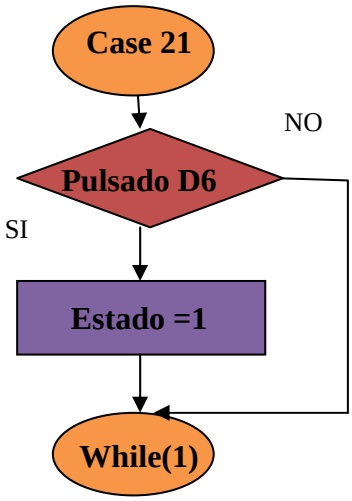
D.- Case 2 (ver y configurar código):

- Se da el valor "**1110 0000**" a la variable mascara ,para que solo parpadeen los leds (7 , 6, 5) conectados al puerto B lo cual indicara el estado de manipulación clave de la caja, para comprenderlo mejor **ver la explicación de la función del timer0.**
- Se comprueba que se ha pulsado el pulsador conectado a D6,utilizamos un bucle While visto anteriormente para evitar problemas.
- Si no se ha pulsado D6 volvemos al bucle While (1)
- Si se ha pulsado D6,almacenamos los 4 bits del código leído en los interruptores (1..4)en la variable **codigo**.Lo almacenamos quedándonos solo los 4bits que nos interresa y haciendo "toggle" a esos bits **(ver explicación en Case 0**).
- Pasamos al siguiente estado en el que se muesta la nueva clave (estado=21)
- Volvemos al bucle While(1)



E.-Case 21 (ver nuevo código)

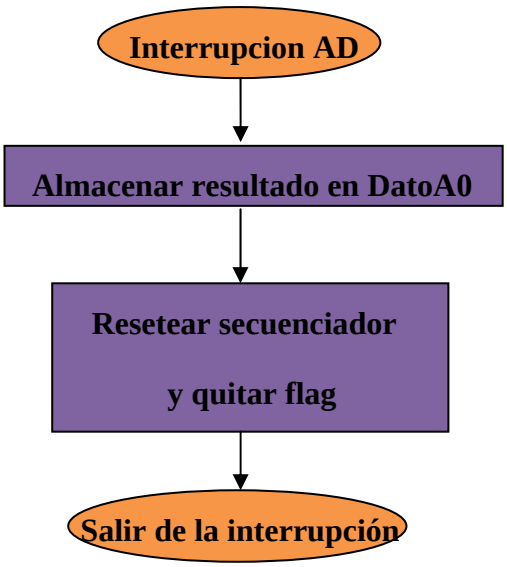
- Comprueba si se pulsa D6,con el bucle while anteriormente visto que evita problemas.
- Si no se ha pulsado D6 vuelve al bucle while(1)
- Si se ha pulsado D6 cambia la variable de estado y pasa a estado 1 (caja abierta)
- Vuelve al bucle while(1)



4.2.2.-Función interrupción AD:

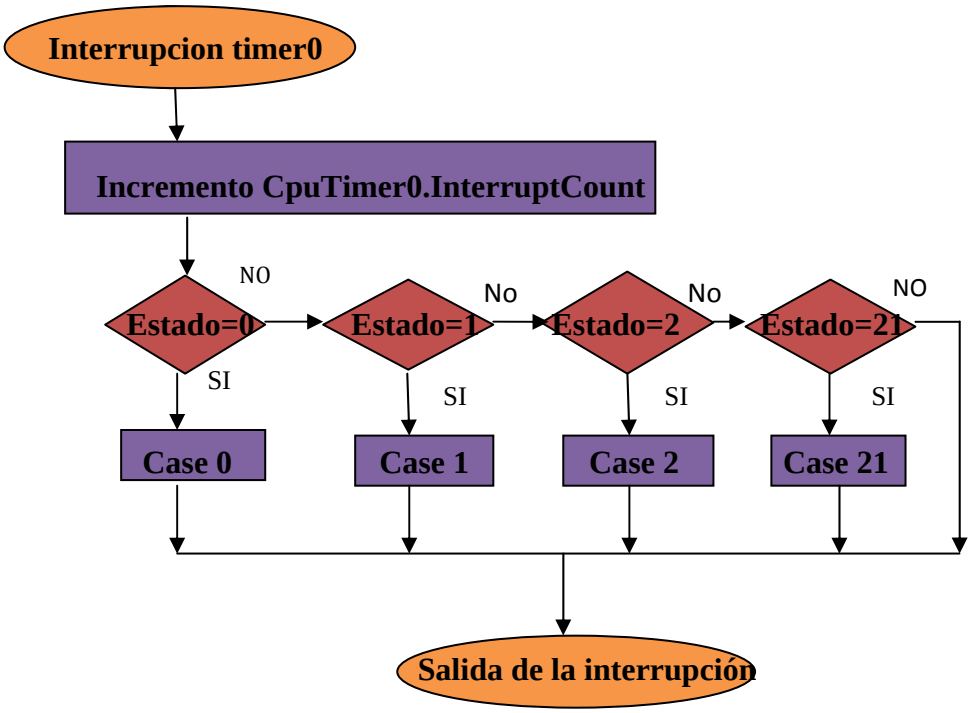
Esta función muestrea el potenciómetro que representa la manipulación de la caja, muestrea el valor AD a una frecuencia de 20Khz empleando el timer2 EVA ,como se ha visto en el apartado de configuración.

Para almacenar el resultado mueve los bits 4 posiciones a la derecha ,ya que el registro de la conversión AD utiliza los 12 bits mas significativos como ya vimos en los ejercicios básicos(**4a**).



4.2.3.- Función interrupción timer0

- Esta función es similar a la principal, esta compuesta por varios casos dependiendo el estado en el que se encuentre.
- La función se ejecuta cada 10us ,temporizados por el timer0
- Incrementamos la variable de la función (CpuTimer0.InterruptCount),para llevar un conteo del tiempo que ha pasado , teniendo en cuenta que se incrementa cada vez que salta la interrupción del timer0, es decir un incremento cada 10us.



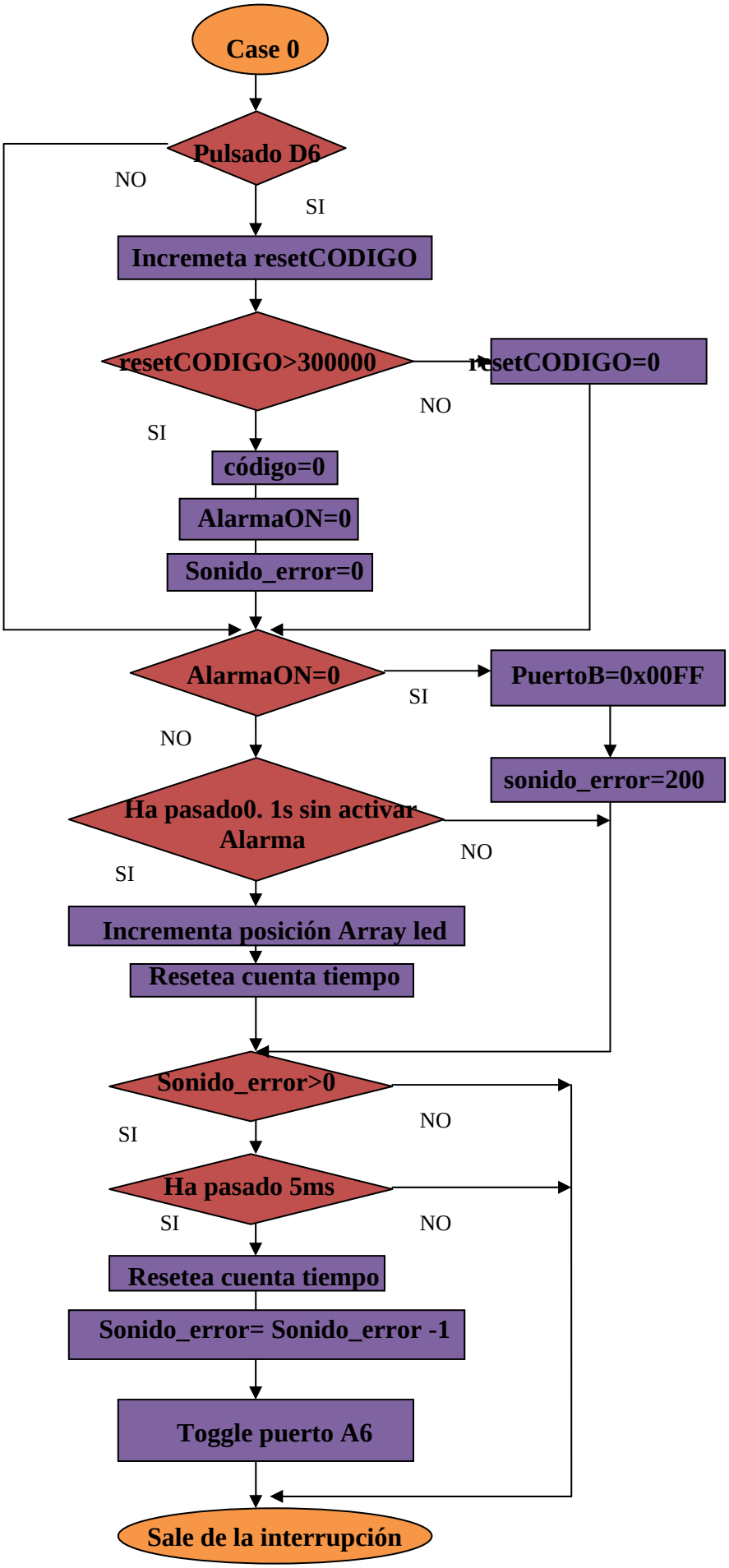
A.-Case 0 timer0:

- Comprueba si se pulsa D6 durante 3s
  - Como la interrupción salta cada 10us ,incrementamos una variable (**resetCODIGO**) cada vez que se introduce en la interrupción y esta pulsando D6.
  - Para saber que ha pasado 3s se contabiliza hasta 300 000 que multiplicado por 10us (tiempo que transcurre entre dos interrupciones),nos da los 3s.
- Si no pasa 3s ,cuando salte la interrupción y no se este pulsando D6 resetea la variable **resetCODIGO .**
- Si pasa 3s
  - Ponemos el código a 0 (código=0)
  - Quitamos la alarma (alarmaON=0)
  - Quitamos el sonido de error (Sonido_error=0)

- Comprobamos si esta activa la alarma
- Si esta activa la alarma
    - Encendemos todos los leds
    - Ponemos la variable Sonido_error=200 ,para emitir el sonido (se vera mas adelante).
- Si no esta activa la alarma ,comprobamos si ha pasado 0.1s sin que la alarma este activa
    - La variable **timestampCOCHE** tiene un valor inicial de 0 y no se incrementa nunca, mientras que la variable **CpuTimer0.InterruptCount** se incrementa una unidad cada 10us (cada interrupción)
    - Cuando la diferencia entre estas variables sea mayor que 10000,habrá pasado 0,1s .

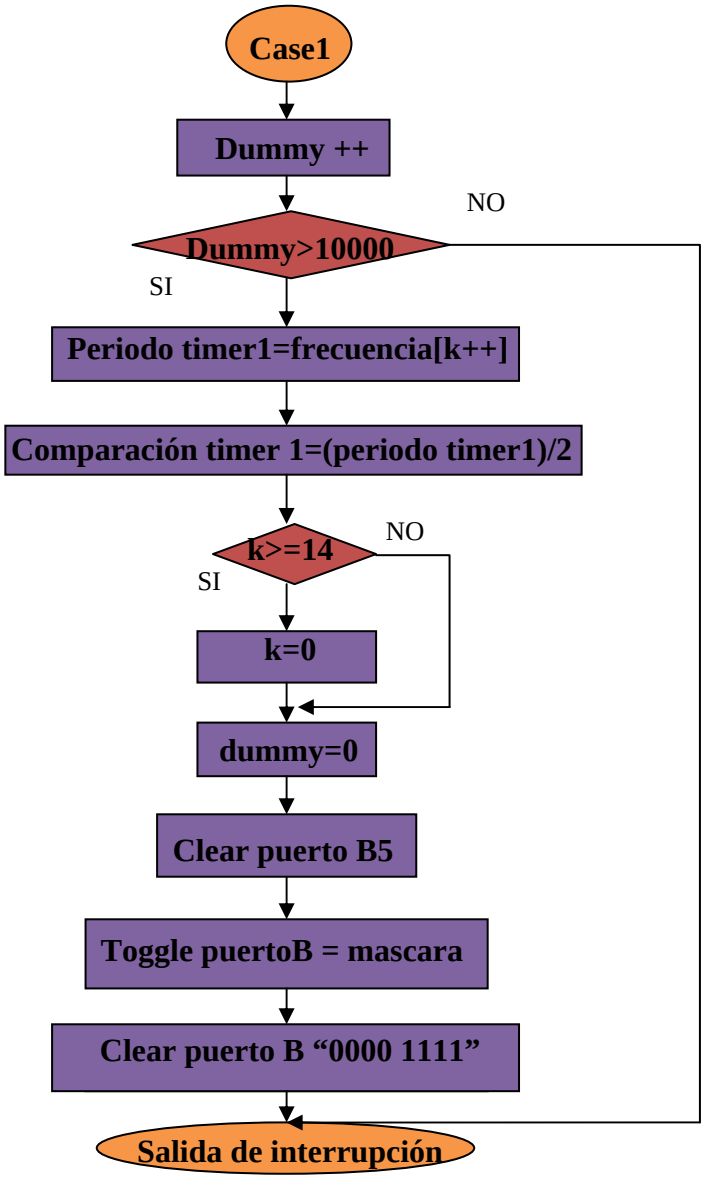
      if ((CpuTimer0.InterruptCount-timestampCOCHE)>10000)

    - Pasado 0,1s se incrementa la posición del array de led
    - Para contar de nuevo 0.1s , se asigna a la variable **timestampCOCHE** el valor de **CpuTimer0.InterruptCount** y se espera a que la diferencia se a 10000 ,asi habrá pasado otro 0.1s
- Comprobar si la variable Sonido_error es mayor que 0,esta variable indica el tiempo que se genera sonido de error, que 206nterr de hacer toggle a la puerta A6
- Si la variable **Sonido_error** es 0 ,se sale de la interrupción
- Si la variable **Sonido_error** es mayor que 0
    - Se comprueba que ha pasado 5ms, esto se hace igual que anteriormente
    - ,se utiliza una variable (**timestampERROR**) ,que no se incrementa ,y se compara con **CpuTimer0.InterruptCount** ,cuando la diferencia sea 500, habrá pasado 5ms.
    - Se asigna a **timestampERROR** el valor de**CpuTimer0.InterruptCount** y se vuelve a esperar que la diferencia se 500(pasa 5ms).
    - Cuando pasa 5ms se hace toggle al puerto A6, de este modo se genera una onda cuadrada (alternando salida valor alto y bajo) de 10ms de periodo y una frecuencia de 100Hz .
    - Pasado 5ms también se decrementa la variable **Sonido_error** ,de modo que cuando sea 0 ya no se hace toggle al puerto y no emite sonido.
    - Mientras que la variable **alarmaON** este a 1 , se va estableciendo la variable **Sonido_error** a 200 ,por lo tanto el sonido no para nunca(hasta que se elimine la alarma.
    - Cando la alarma este desactivada y se coloque una contraseña errónea, se carga la variable **Sonido_error** a 200,y esta va decrementando hasta llegar a 0,el sonido que se emite es corto, ya que no se vuelte a incrementar la variable **Sonido_error** (case 0 del bucle while).
- Una vez comprobado las anteriores condiciones ,se sale de la Interrupción


Case 0
Pulsado D6
NO
SI
Incremeta resetCODIGO
resetCODIGO>300000
resetCODIGO=0
NO
SI
código=0
AlarmaON=0
Sonido_error=0
AlarmaON=0
SI
PuertoB=0x00FF
NO
sonido_error=200
Ha pasado0. 1s sin activar Alarma
NO
SI
Incrementa posición Array led
Resetea cuenta tiempo
Sonido_error>0
NO
SI
Ha pasado 5ms
SI
NO
Resetea cuenta tiempo
Sonido_error= Sonido_error -1
Toggle puerto A6
Sale de la interrupción

B.-Case 1 timer0

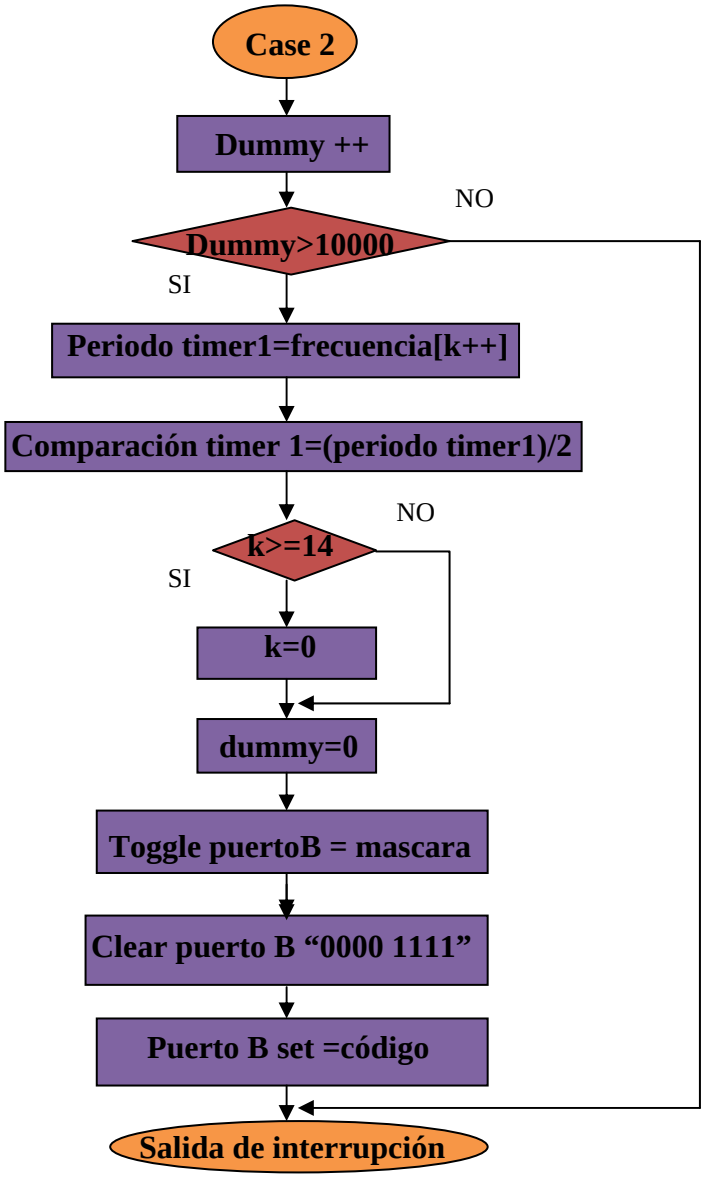
- Incrementamos la variable dummy
- Si la variable dummy llega al valor 10 000 habrá pasado 0.1s (1 incremento por interrupción, una interrupción cada 10us)
- Cuando pasa 0,1s
    - Colocamos la siguiente posición del array frecuencia, en el periodo del timer1 EVA ,es decir cambiamos la frecuencia
    - Colocamos (periodo/2) ,en el valor de comparación
    - Variando la frecuencia cada 0.1s generamos la melodia
    - Ponemos a 0 la variable dummy para volver a temporizar 0.1s
    - Hacemos clear en el puerto B5, ya que este led quedara encendido si venimos del Case21
    - Hacemos toggle al puerto B utilizando la mascara ,asi dependiendo del estado parapadean 2 o 3 leds (el parapadeo se produce al cambiar el estado del led cada 0,1s)
    - Hacemos clear “0000 1111” al puerto B a los 4 bits menos significativos ,ya que estos muestran la clave en los estados 2 y 21 ,y pueden quedar activos si venimos de esos estados.



C.-Case 2 timer0

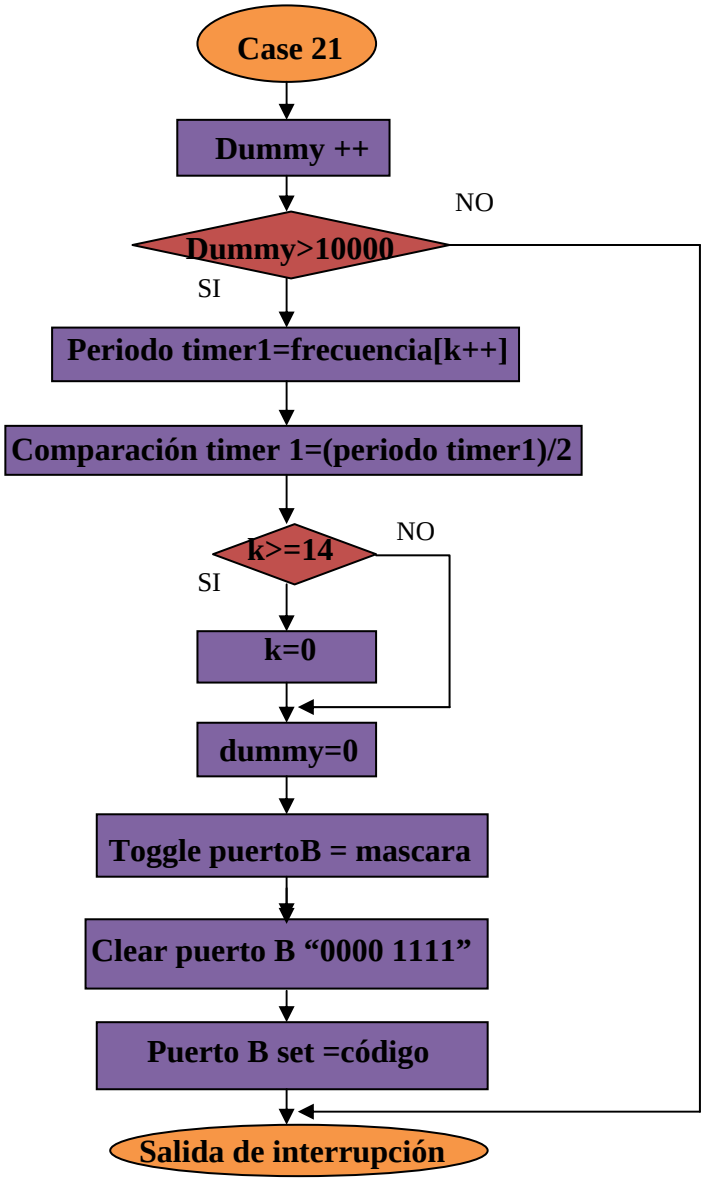
Este caso es similar al "Case 1" ,ya que la melodía se temporiza y varia igual que en el caso anterior ,la única diferencia residen en el encendido de los leds.

- Incrementamos la variable dummy
- Si la variable dummy llega al valor 10 000 habrá pasado 0.1s (1 incremento por interrupción, una interrupción cada 10us)
- Cuando pasa 0,1s
    - Colocamos la siguiente posición del array frecuencia, en el periodo del timer1 EVA ,es decir cambiamos la frecuencia
    - Colocamos (periodo/2) ,en el valor de comparación
    - Variando la frecuencia cada 0.1s generamos la melodia
    - Ponemos a 0 la variable dummy para volver a temporizar 0.1s
    - Hacemos toggle al puerto B utilizando la mascara ,asi dependiendo del estado parapadean 2 o 3 leds (el parapadeo se produce al cambiar el estado del led cada 0,1s)
    - Hacemos clear "0000 1111" al puerto B a los 4 bits menos significativos ,ya que estos pueden muestrar la clave anterior.
    - Hacemos "set" puerto B y mostramos el código actual.



D.-Case 21 timer0

Este caso es exactamente igual que el anterior, con el mismo código,ya que la melodía y encendido de los leds son los mismos.
La función timer0 que controla la melodía y encendido de los leds es igual,sin embargo existe diferencia entre el Case 2 y Case21 en la función principal.
El Case 2 se encarga de mostrar la clave y registrar la nueva en caso que se pulse D6, y el Case 21 muestra la nueva clave y espera a que se pulse D6 para volver a caja abierta.



4.3.- CODIGO COMENTADO

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"

// CAJA DE MUSICA CON CÓDIGO DE APERTURA

interrupt void CPUTimer0_Interrupt(void);//Funcion interrupcion timer0
interrupt void ADC_Interrupt(void);//Funcion interrupcion Convertidor AD

int sonido_error=0;//tiempo que debe estar activo  el sonido error

int i=0;//para recorrer el array de luces
int k=0;//para recorrer el array de frecuencias
int codigo=0;//codigo que abre la caja
int codigo_posible=0;//codigo que se introduce por los interruptores
int mascara=0; //mascara para el parpadeo de leds
long j=0; //conteo para asegurar que se registra bien el estado del pulsador
long timestampCOCHE=0; //Para temporizar secuencia luces caja cerrada
long timestampERROR=0; //Para emitir el sonido de error
long resetCODIGO=0;//para el conteo de 3s y hacer reset posterior
int DatoA0=0;//tension potenciometro indica manipulacion de la caja
int umbralA0=2000;//umbral para activar alarma
int alarmaON=0; //1.alarma activada 0.alarma desactivada
int dummy=0;//para temporizar 0.1s y cambiar a la siguiente posicion del array de luces
int led[14]={1,2,4,8,16,32,64,128,64,32,16,8,4,2};//secuencia de luces caja cerrada
int frecuencia[15]={2219,1973,1776,1665,1480,1332,1184,1110,
1184,1332,1480,1665,1776,1973,2219};//melodia caja abierta
int botonD1,botonD6;//registra pulsacion D1 apertura caja y D6 cambio clave
int estado=0;//0:caja cerrada,1:caja abierta,2:Ver y conf clave,21:ver nueva clave
void main()
{

    InitSysCtrl();

    EALLOW;
    GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=0; //puerta GPIOA6 como salida
    GpioMuxRegs.GPADIR.bit.GPIOA6=1;

    GpioMuxRegs.GPBMUX.all=0x0000;//puertoB salida (leds)
    GpioMuxRegs.GPBDIR.all=0x00FF;

    GpioMuxRegs.GPDMUX.bit.T2CTRIP_SOCA_GPIOD1=0; //puerta GPIOD1 como entrada(pulsador)
    GpioMuxRegs.GPDDIR.bit.GPIOD1=0;

    GpioMuxRegs.GPDMUX.bit.T4CTRIP_SOCB_GPIOD6=0;//puerta GPIOD6 como entrada(pulsador)
    GpioMuxRegs.GPDDIR.bit.GPIOD6=0;
    EDIS;
    GpioDataRegs.GPBCLEAR.all=0x00FF;

    InitPieCtrl();
    InitPieVectTable();


    EALLOW;
    PieVectTable.TINT0=&CPUTimer0_Interrupt; //vector interrupcion timer0
    PieVectTable.ADCINT=&ADC_Interrupt;  //vectror interrupcion AD
    EDIS;
    PieCtrlRegs.PIEIER1.bit.INTx7=1;  //Interrupción de CpuTimer0(grupo7 linea int1)
    PieCtrlRegs.PIEIER1.bit.INTx6=1;  // Interrupción ADC (grupo6 linea int1)
    IER|=0x0001; //habilita linea int1 de interrupciones
    EINT;
```

```
//configuracion AD
InitAdc();

AdcRegs.ADCTRL1.bit.SEQ_CASC=0; //modo dual
AdcRegs.ADCTRL1.bit.CONT_RUN=0; //parar al final de la secuencia
AdcRegs.ADCTRL1.bit.CPS=0;       //prescaler de la conversion 1/1
AdcRegs.ADCMAXCONV.all =0x0000;  //(nº conversiones automaticas+1) //0+1
AdcRegs.ADCCHSELSEQ1.bit.CONV00=0x0;//pone a 0 reg control secuencia conversion
AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1=1; //conversiones son iniciadas por EVA(trigger)
AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1=1;  //habilita el secuenciador 1
AdcRegs.ADCTRL3.bit.ADCCLKPS=2;      //preescaler del reloj ADC 1/4

EvaRegs.GPTCONA.bit.T2TOADC=2;//Conversion AD activada por timer2
EvaRegs.GPTCONA.bit.TCMPOE=1; //habilita registro por comparacion

 //configuracion timer1
 //habilita timer1 por comparacion, preescala 1/128, continuamente ascendente
 EvaRegs.T1CON.all=0x1702;
 EvaRegs.GPTCONA.bit.T1PIN=1;  //activo a nivel bajo

 //configuracion timer2
 //modo continuamente ascendente,sin preescaler y funcionamiento por comparacion
 EvaRegs.T2CON.all=0x1002;
 EvaRegs.T2PR = 3750; //periodo timer2 para muestreo AD a 20Khz
 EvaRegs.T2CON.bit.TENABLE=1;//habilita timer2 empieza a contar

 InitCpuTimers();
 ConfigCpuTimer(&CpuTimer0,150,10);//configura timer0 salta interrupcion cada 10us
 CpuTimer0Regs.TCR.bit.TSS = 0; //habilita timer0 empieza a contar

 while(1)
 {
     switch(estado)
     {
         case 0: //Caja cerrada. Esperar a pulsar el bótónD1 para abrirla
         {
             if (DatoA0>=umbralA0) //Si muestra AD supera umbral activa alarma
             {
                 alarmaON=1;
             }
             botonD1=GpioDataRegs.GPDDAT.bit.GPIOD1;//comprueba si se ha pulsado D1
             if(botonD1==0)
             {
                 while(botonD1==0)
                 {
                     for(j=0;j<500000;j++);
                     botonD1=GpioDataRegs.GPDDAT.bit.GPIOD1;
                 }
                 //Comprueba codigo
                 codigo_posible=15-((GpioDataRegs.GPBDAT.all>>8) & 0x0F);
                 //si codigo es correcto
                 if ((codigo==codigo_posible) )

                 {
                 GpioDataRegs.GPBCLEAR.all=0x00FF;
                 EALLOW;
                 GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=1;//activa func primaria A6
                 EDIS;
                 EvaRegs.T1CON.bit.TENABLE=1;  //activa timer1 EVA
                 EvaRegs.T2CON.bit.TENABLE=0;  //para timer2 EVA
                 alarmaON=0;             //quita alarma
                 estado=1;           //pasa a estado caja abierta
                 }
                  else//si codigo no es correcto genera leve pitido
                  {
                      timestampERROR=CpuTimer0.InterruptCount;
                      sonido_error=200;
                  }
              }
              break;
          }
```

```
            case 1://Caja abierta.Esperar a pulsar el botónD1 o botonD6
            {
                mascara=0x00C0;//para el parpadeo de los leds
                botonD6=GpioDataRegs.GPDDAT.bit.GPIOD6;//comprueba si se ha pulsado D6
                if(botonD6==0)
                {
                    while(botonD6==0)
                    {
                        for(j=0;j<500000;j++);
                        botonD6=GpioDataRegs.GPDDAT.bit.GPIOD6;
                    }
                    estado=2;//si se ha pulsado D6 cambia al estado 2
                    break;
                }
                //comprueba si se pulsa boton de cierre D1
                botonD1=GpioDataRegs.GPDDAT.bit.GPIOD1;
                if(botonD1==0)
                {
                    while(botonD1==0)
                    {
                        for(j=0;j<500000;j++);
                        botonD1=GpioDataRegs.GPDDAT.bit.GPIOD1;
                    }
                    //si se pulsa D1
                    EALLOW;
                    GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=0;//quita funcion primaria A6
                    GpioMuxRegs.GPADIR.bit.GPIOA6=1;
                    EDIS;
                    EvaRegs.T1CON.bit.TENABLE=0;  //para timer1 EVA
                    EvaRegs.T2CON.bit.TENABLE=1;  //activa timer2 EVA
                    sonido_error=0;
                    estado=0;
                }
                break;
            }
            case 2: //ver y configurar codigo.Modificar switches y esperar a
                    //pulsar botonD6 para almacenar nuevo código.
            {
                mascara=0x00E0;//para parpadeo de leds
                botonD6=GpioDataRegs.GPDDAT.bit.GPIOD6;//comprueba si se pulsa D6
                if (botonD6==0)
                {
                    while(botonD6==0)
                    {
                        for(j=0;j<500000;j++);
                        botonD6=GpioDataRegs.GPDDAT.bit.GPIOD6;
                    }//si se pulsa D6 almacena nuevo codigo
                    codigo=15-((GpioDataRegs.GPBDAT.all>>8) & 0x0F);
                    estado=21;//pasa a siguiente estado
                }
                break;
            }
            case 21: // ver nuevo codigo
                    //si se pulsa D6 pasa a caja abierta
            {
                botonD6=GpioDataRegs.GPDDAT.bit.GPIOD6;
                if (botonD6==0)
                {
                    while(botonD6==0)
                    {
                        for(j=0;j<500000;j++);
                        botonD6=GpioDataRegs.GPDDAT.bit.GPIOD6;
                    }
                    estado=1;
                }
                break;
            }
        }
    }
}
```

```
interrupt void ADC_Interrupt(void)
{  //almacena resultado muestra AD en DatoA0
    DatoA0=AdcRegs.ADCRESULT0>>4;
    AdcRegs.ADCTRL2.bit.RST_SEQ1=1;
    AdcRegs.ADCST.bit.INT_SEQ1_CLR=1;
    PieCtrlRegs.PIEACK.bit.ACK1=PIEACK_GROUP1;
}

//funcion interrupcion timer0 (cada 10us)
interrupt void CPUTimer0_Interrupt(void)
{
    CpuTimer0.InterruptCount++; //incrementa variable para conteo de tiempo
    switch (estado)
    {
        case 0:
        {//comprueba que se pulsa D6 durante 3s
            if (GpioDataRegs.GPDDAT.bit.GPIOD6==0)
            {
                resetCODIGO++;
                if (resetCODIGO>300000)
                {   //si pasa 3s con D6 pulsado resetea la caja
                    codigo=0;
                    alarmaON=0;
                    sonido_error=0;
                }
            }
            else //si no pasa 3s vuelve a 0 para contar de nuevo
            {
                resetCODIGO=0;
            }

           if (alarmaON==0) //comprueba si se ha activado la alarma
           {     //si no se activa cada segundo incrementa una posicion de array led
               if ((CpuTimer0.InterruptCount-timestampCOCHE)>10000)
               {
                   timestampCOCHE=CpuTimer0.InterruptCount;
                   GpioDataRegs.GPBDAT.all=led[i++];
                   if (i==14)
                   {
                       i=0;
                   }
               }
           }
           else  //si se activa alarma carga la variable sonido_error
                 //que es el timepo que emite el sonido
           {
               GpioDataRegs.GPBDAT.all=0x00FF;
               sonido_error=200;
           }
           if (sonido_error>0) //si la variable sonido_error tiene un valor >0
           {                   //emite sonido por puerto A6
               if((CpuTimer0.InterruptCount-timestampERROR)>500)
               {   //espera que pase 5ms
               timestampERROR=CpuTimer0.InterruptCount;
               sonido_error=sonido_error-1;
               GpioDataRegs.GPATOGGLE.bit.GPIOA6=1;//alterna  0 y 1 produce sonido
               }
           }

           break;
       }
```

```
        case 1:
        {
            dummy++;
            if (dummy>10000) //comprueba que ha pasado 0.1s y si es asi
                             //varia la frecuencia que se emite por A6
            {
                EvaRegs.T1PR=frecuencia[k++];
                EvaRegs.T1CMPR=EvaRegs.T1PR/2;
                if(k>=14)
                {
                    k=0;
                }
                dummy=0;
                GpioDataRegs.GPBCLEAR.bit.GPIOB5=1;
                GpioDataRegs.GPBTOGGLE.all=mascara;//parpadeo cada 0,1s
                GpioDataRegs.GPBCLEAR.all=0xF;
            }
            break;
        }
        case 2:
        case 21:
        {
            dummy++;
            if (dummy>10000)//comprueba que ha pasado 0.1s y si es asi
            {                //varia la frecuencia que se emite por A6
                EvaRegs.T1PR=frecuencia[k++];
                EvaRegs.T1CMPR=EvaRegs.T1PR/2;
                if(k>=14)
                {
                    k=0;
                }
                dummy=0;
                GpioDataRegs.GPBTOGGLE.all=mascara; //parpadeo cada 0.1s
                GpioDataRegs.GPBCLEAR.all=0xF;
                GpioDataRegs.GPBSET.all=codigo; //muestra codigo
            }
            break;
        }
    }
    PieCtrlRegs.PIEACK.bit.ACK1=PIEACK_GROUP1;//sale de la interrupcion
}
```

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

5.-CONCLUSIONES Y FUTUROS TRABAJOS

Con este proyecto se ha tratado de explicar de una forma práctica que es un DSP en general y el F2812 en particular.

En un primer punto hemos descrito las características principales del DSP F2812,con el objetivo de comprender el funcionamiento de sus módulos, tales como los temporizadores y las interrupciones, generación de ondas PWM mediante el uso del Event Manager, el uso del convertidor A/D etc.

En un segundo punto hemos expuesto el diseño de nuestra placa de expansión, la cual utilizaremos para la puesta en practica de nuestras aplicaciones, en este apartado hemos desarrollado los cálculos necesarios para no causar daños a ningún componente y que todo funcione según lo previsto, además de cálculos hemos comprobado el diseño por módulos en una protoboard antes de crear el diseño final en el cual hemos utilizado una placa de doble cara que ha sido fabricada mediante fresadora.

Hemos de señalar que la placa del presente proyecto es una versión 1.0, en la cual el objetivo era obtener un soporte para comprobar el correcto funcionamiento de los ejercicios programados y para poder ver el resultado de nuestra programación. Como futuro trabajo dejamos la creación de una versión 2.0 en la cual reduciremos el tamaño de la placa, utilizando componentes smd .

El tercer punto que hemos desarrollado es la programación de aplicaciones sencillas para la compresión de cada uno de los módulos explicados en el punto 1, estos módulos son temporizadores, interrupciones, Event Manager y convertidor AD.

A todos los ejercicios realizados le hemos añadido una explicación detallada, además de un flujograma y el código comentado al detalle, con la explicación de las características de los módulos del punto 1 y el ejemplo práctico del punto 3,el alumno tendrá un apoyo donde resolver sus dudas en futuras aplicaciones.

En el punto cuarto hemos desarrollado una Aplicación que utiliza todos los modulos vistos anteriormente, esta aplicación es mas compleja y es una sinteis de todo lo visto en el punto 3.

Como conclusión ,el desarrollo de aplicaciones básicas en el F2812 que pueden ser visualizadas en la placa de expansión que hemos desarrollado, sirve de base solida para poder realizar futuras aplicaciones con este DSP de Texas Instruments o con otros similares o más complejos , ya que hemos conseguido una buena base formativa para seguir aprendiendo y desarrollando aplicaciones mas complejas.

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE

6.-BIBLIOGRAFIA

- Apuntes de clase de la asignatura “diseño y simulación electrónica” ,de ITI, Esp Electrónica Industrial.

- CD “C2000 Teaching materials” de Texas Instruments

- “Digital Signal Processors Data Manual” de Texas Instruments, Abril 2001 revisado diciembre 2004.

- Spectrum digital eZdsp F2812 datasheets.

- www.alldatasheet.es para components discretos.

ESTA PAGINA ESTA EN BLANCO INTENCIONADAMENTE